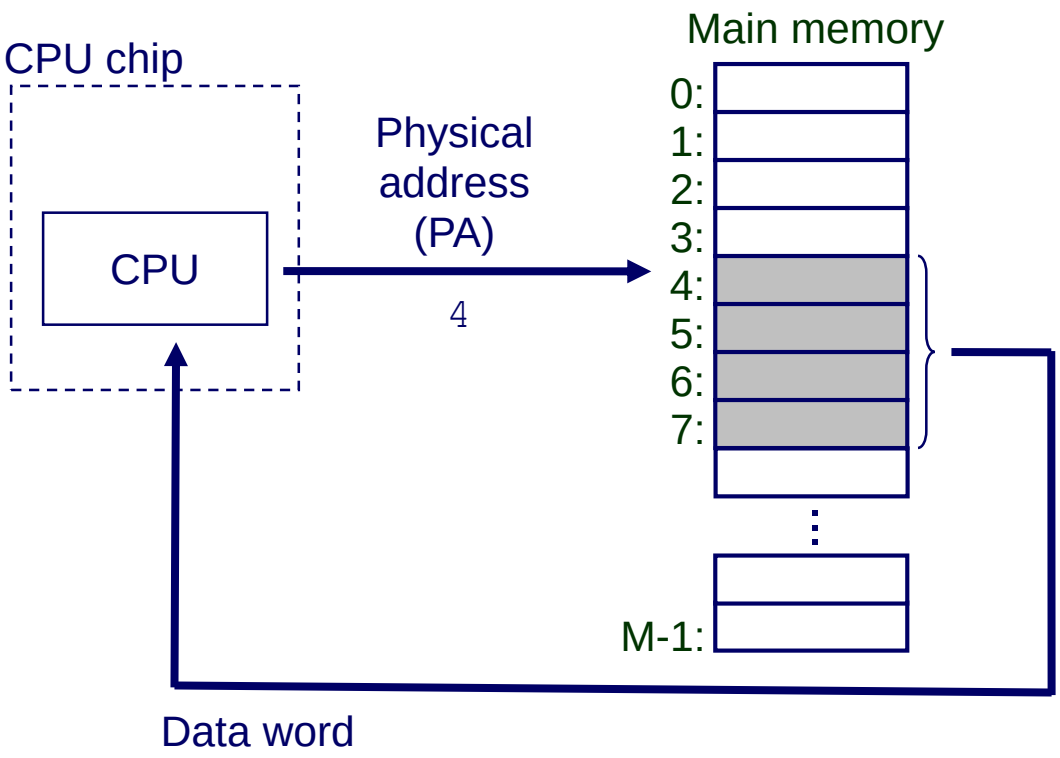
虚拟内存

2021年秋

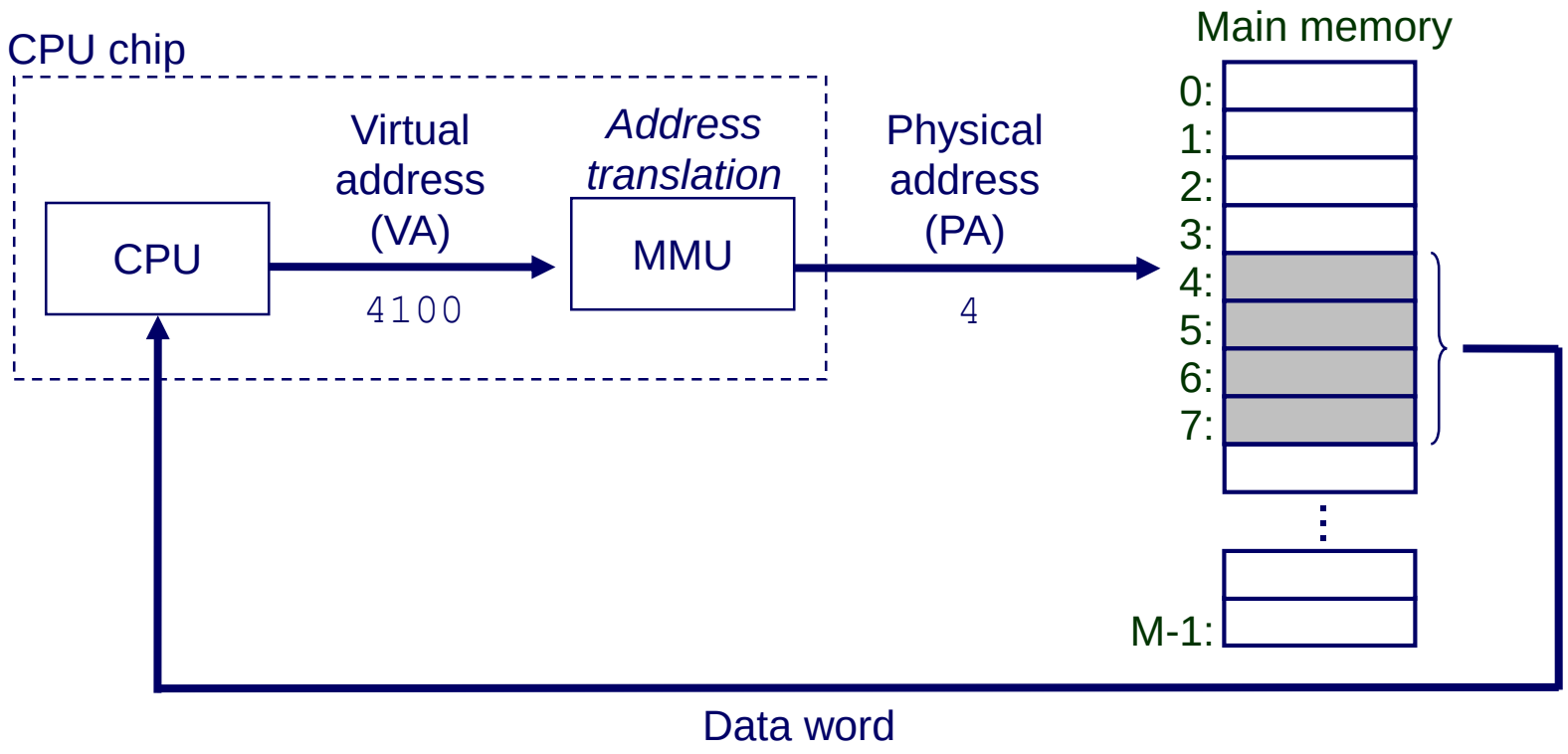
内容提要

- 为什么需要虚拟内存？
- 页式内存管理
  - TLB
- 例子：RISC-V 页表管理
- 段式内存管理与段页式内存管理
- 例子：x86 页表管理

物理内存访问


CPU chip
CPU
Physical address (PA)
4
Main memory
0:
1:
2:
3:
4:
5:
6:
7:
M-1:
Data word

虚拟内存访问



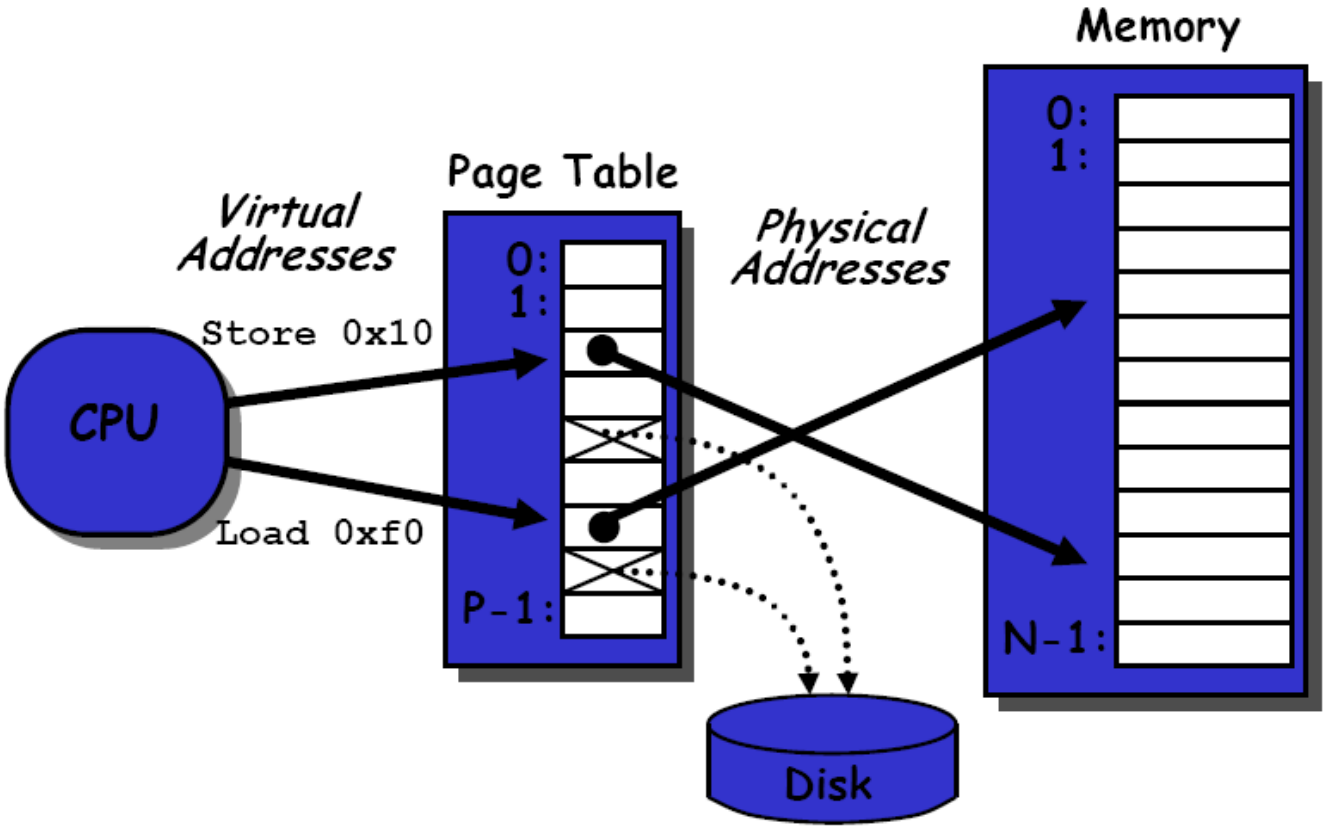
- 虚拟地址（逻辑地址）：程序员编程使用的地址
- 物理地址（实地址）：物理存储器的地址

两个问题

- 如果 程序的工作集大小 大于 物理内存大小，程序还能执行吗？
  - 缓存的思想（虚拟化）

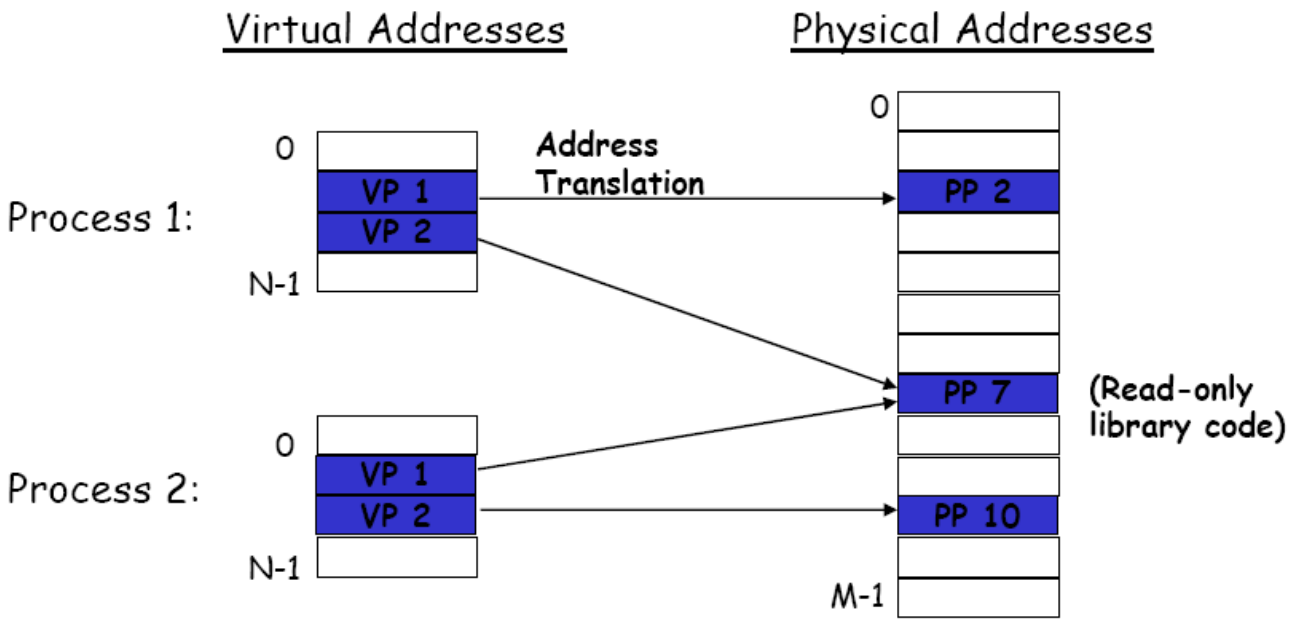
- 如果多个程序共享使用，那么
  - 程序员在编程时，怎么知道在哪儿分配内存？
  - 多个程序需要共享时，怎么办？
  - 多个程序同时执行时，某进程（程序）不想让另一个进程看到或者修改本进程的内容，怎么办？
  - 思想：程序员编程使用的空间与程序运行空间相互独立

（1）独立的逻辑地址空间



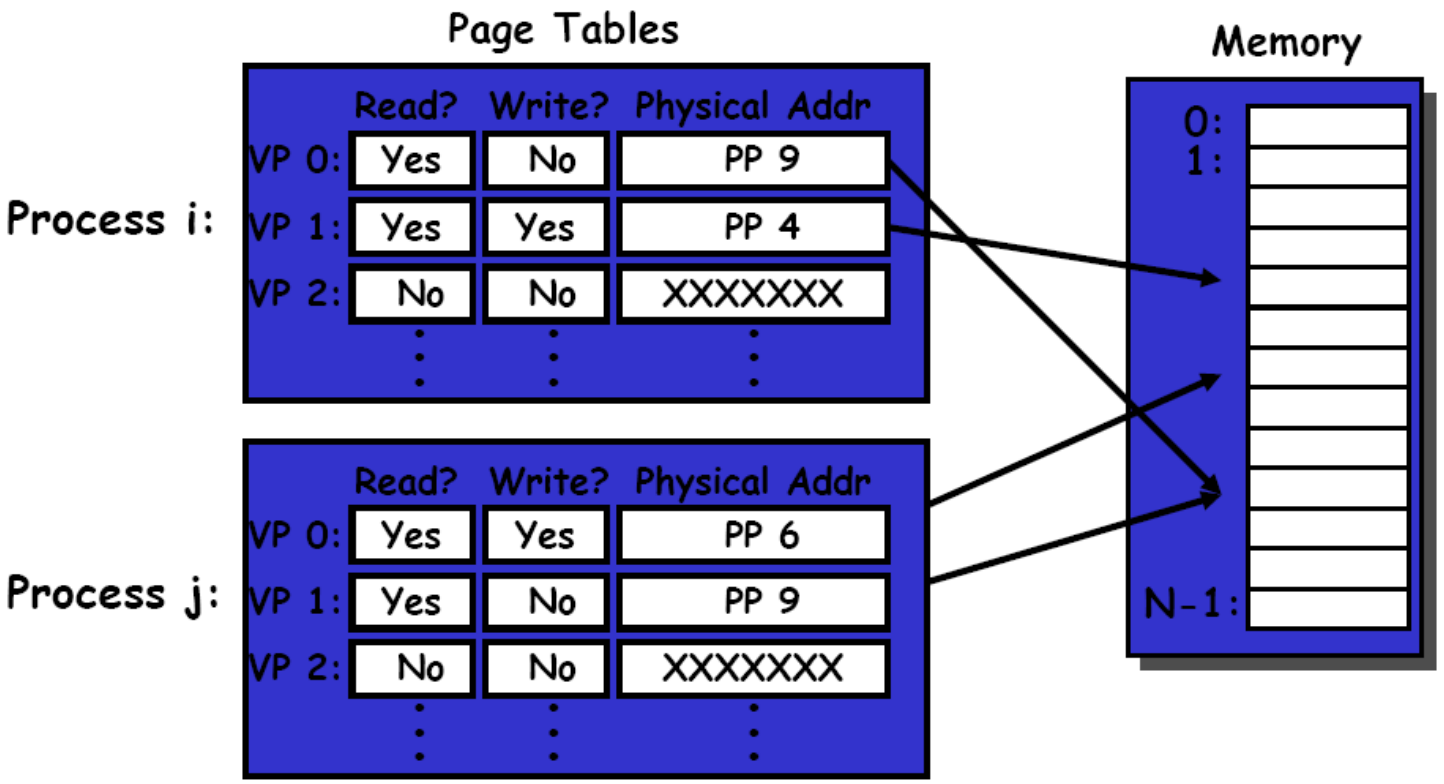
- 通过页表将虚地址转换为实地址

（2）实现内存共享



- 每个进程有独立的逻辑地址空间
- 建立逻辑地址和物理地址的转换机制

（3）实现内存的保护



- 页表中存放有访问权限
  - 通过硬件来保证权限（操作系统的“陷阱”操作）

虚拟存储器的目的

- 容量
  - 获得运行比物理存储器更大空间程序的能力
- 存储管理
  - 内存的分配以及虚实地址转换
- 保护
  - 操作系统可以对虚拟存储空间进行特定的保护. . .
- 灵活
  - 程序的某部分可以装入主存的任意位置
- 提高存储效率
  - 只在主存储器中保留最重要的部分
- 提高并行度
  - 在进行段页替换的同时可以执行其它进程
- 可扩展
  - 为对象提供了扩展空间的能力.

虚拟内存与高速缓存的比较

	虚拟内存	**高速缓存**
层次	“内存-外存”层次	“CPU缓存-内存”层次
主要目的	解决存储容量的问题	解决主存储器与CPU性能的差距
数据交换粒度与频次	单位时间内数据交换次数较少，但每次交换的数据量大，达几十至几千字节	单位时间内数据交换的次数较多，每次交换的数据量较小，只有几个到几十个字节
实现主体	主要由操作系统管理	由硬件实现

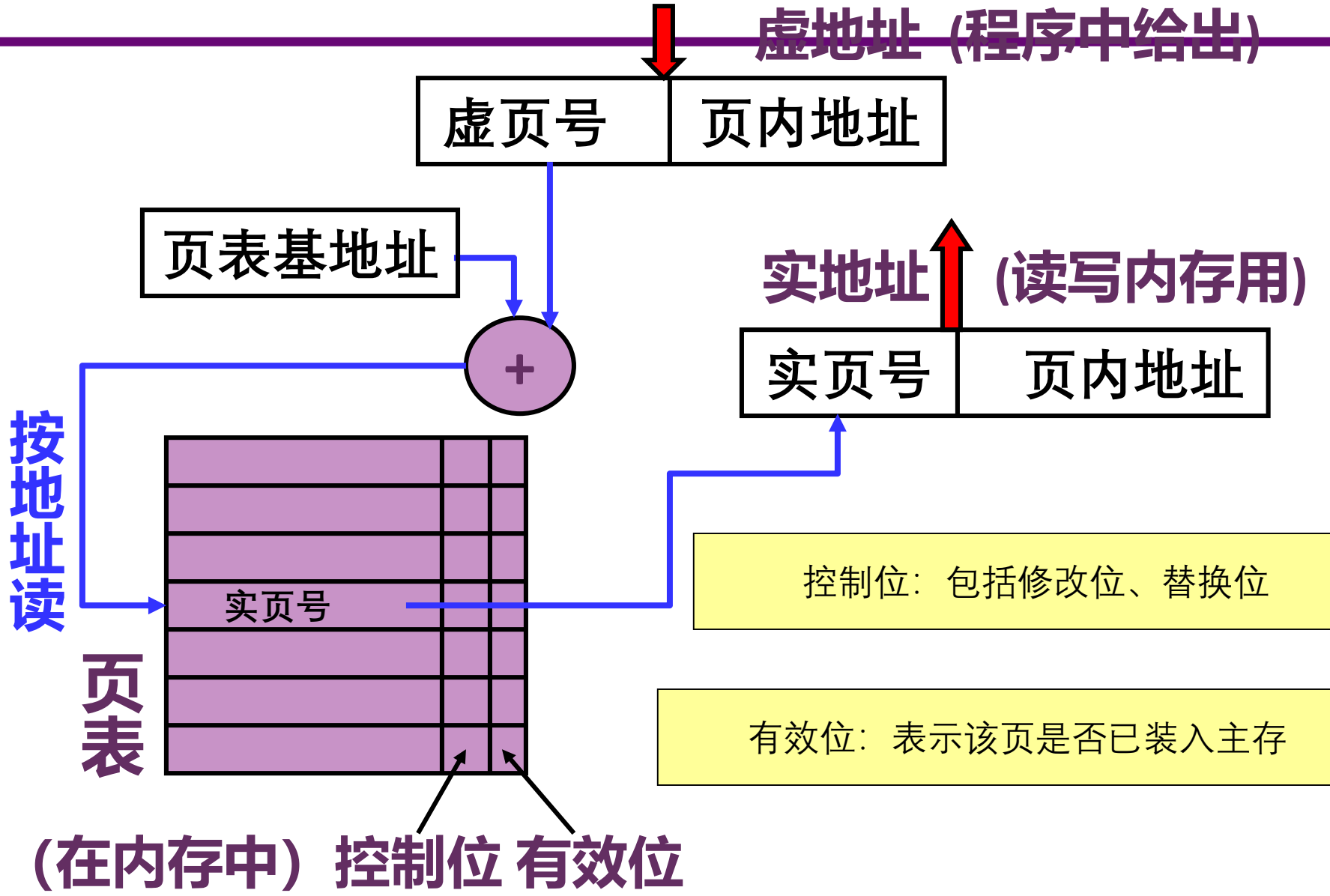
内容提要

- 为什么需要虚拟内存？
- 页式内存管理
  - TLB
- 例子：RISC-V 页表管理
- 段式内存管理与段页式内存管理
- 例子：x86 页表管理

页式存储管理

- 将主存和虚存划分为固定大小的页
- 以页为单位进行管理和数据交换
- 虚地址=虚页号+页内地址
- 实地址=实页号+页内地址
- 通过页表进行管理
  - 页表基地址寄存器
  - 实页号
  - 控制位

页表内容和页式管理


虚地址（程序中给出）
虚页号
页内地址
页表基地址
+
实地址
(读写内存用)
实页号
页内地址
按地址读
实页号
页表
控制位：包括修改位、替换位
有效位：表示该页是否已装入主存
(在内存中) 控制位 有效位

页表大小

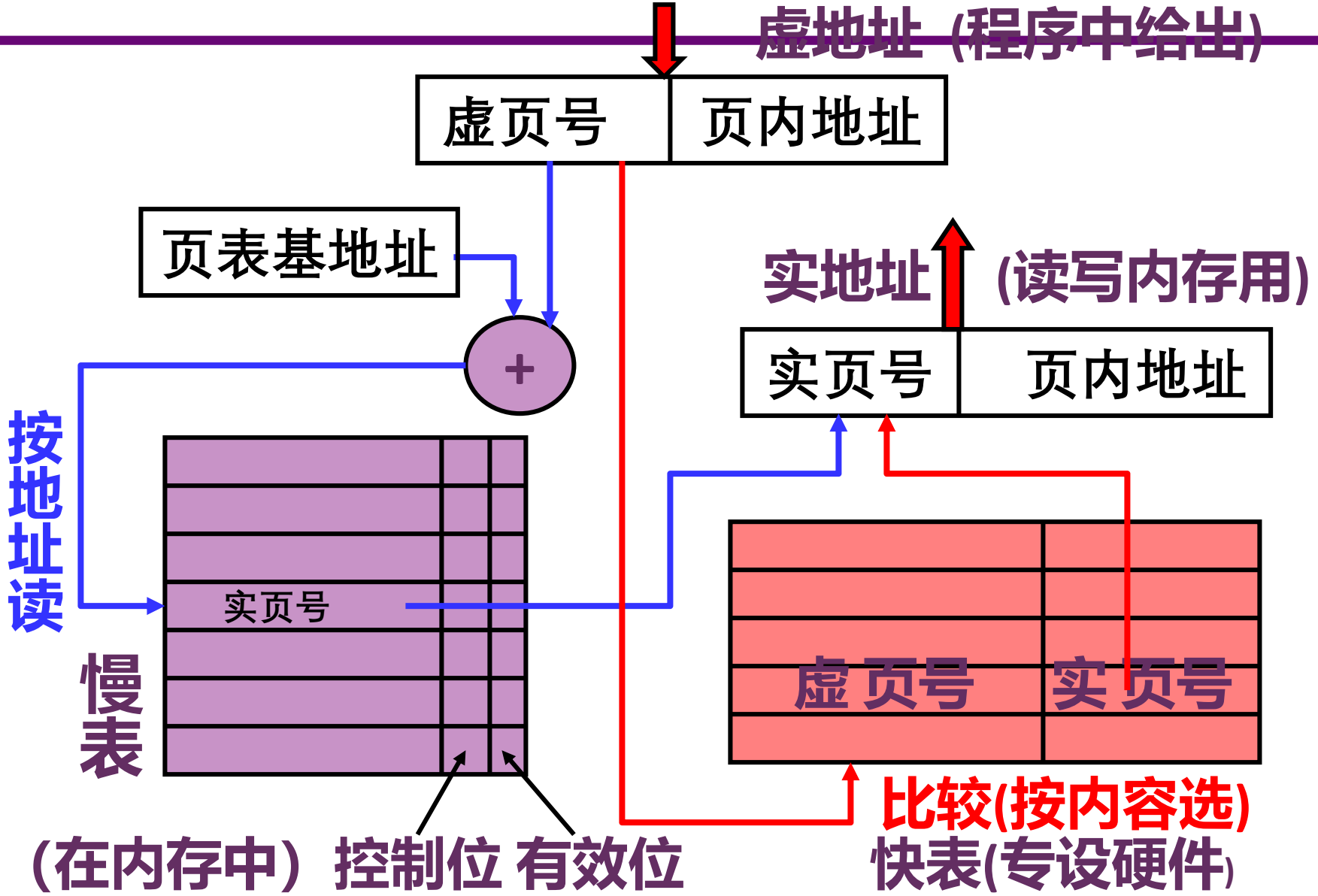
- 与虚页数直接相关，但是
  - 虽然理论上每个进程的逻辑空间很大，但其实大部分应该是不活跃的
  - 实际调入到内存的内容不可能超过物理存储空间
- 如何减少页表本身所占的空间？
  - 而且还要实现简单
    - 页表访问频繁
- 两种途径
  - 层次页表（hierarchical page table）
  - 反转页表（inverted page table）

页式虚拟存储器的访问过程

- 1. 得到程序给出的虚地址；
- 2. 由虚地址得到虚页号；
- 3. 访问页表，得到对应的实页号；
- 4. 若该页已在内存中，则根据实页号得到实地址，访问内存；
- 5. 否则，启动输入输出系统，读出对应页装入主存，再进行访问。

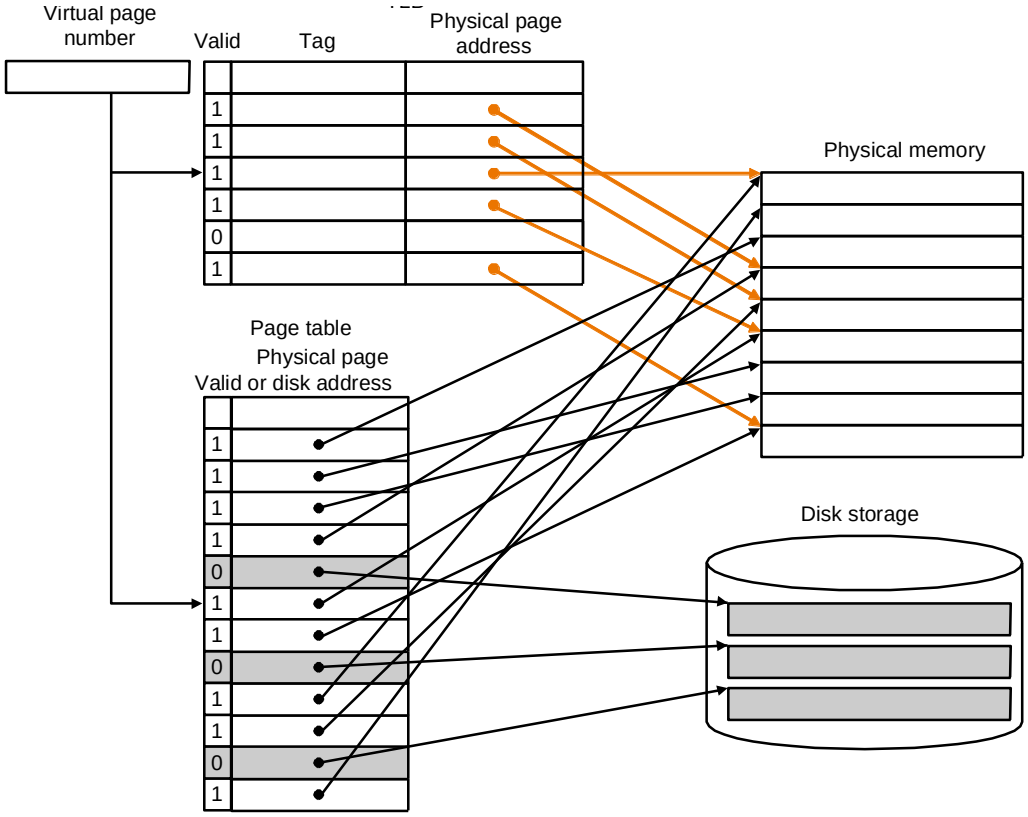
增加由硬件实现的快表，提高访问速度

页表内容和页式管理


虚地址 (程序中给出)
虚页号
页内地址
页表基地址
+
实地址
(读写内存用)
实页号
页内地址
按地址读
实页号
慢表
虚 页号
实 页号
比较(按内容选)
(在内存中) 控制位 有效位
快表(专设硬件)

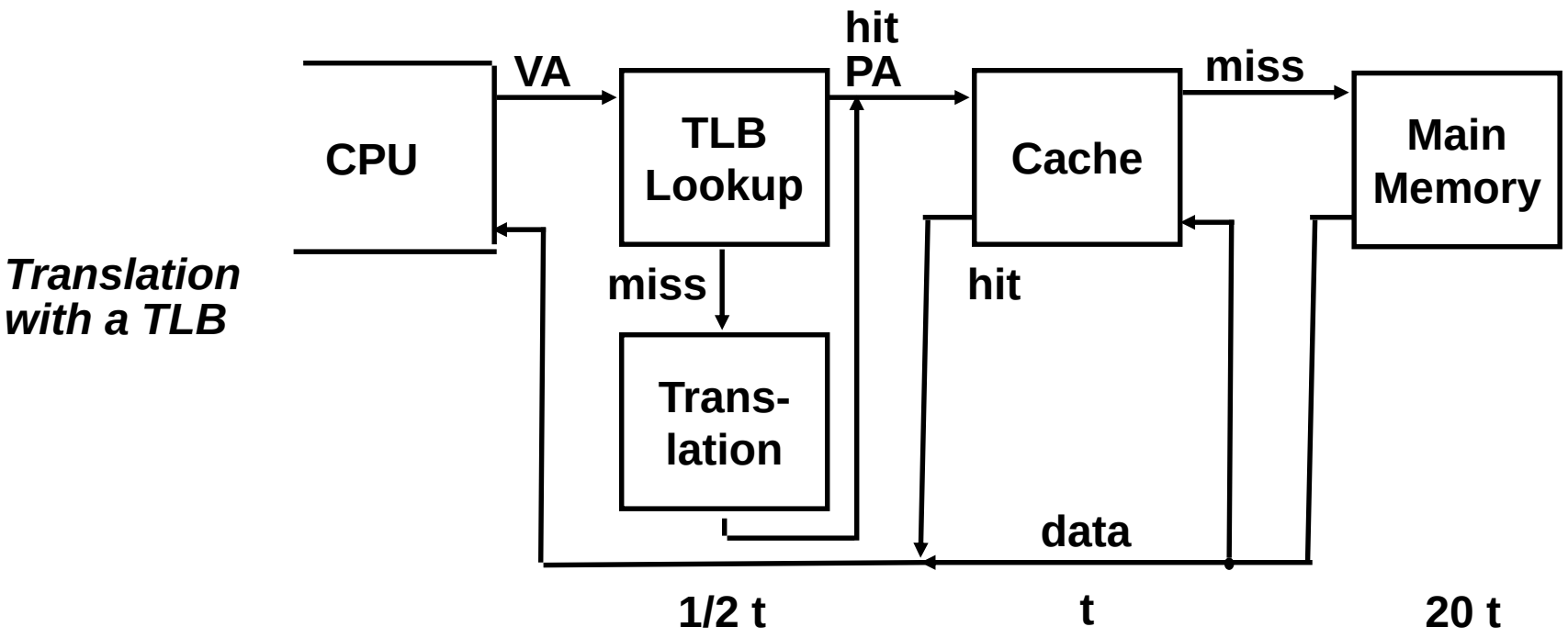
转换旁路缓冲（TLB）

- 访问频繁:速度是第一位的
- TLB 缺失将造成：
  - 流水线停止
  - 通知操作系统
  - 读页表
  - 将表项写入 TLB
  - 返回到用户程序
  - 重新访问
- 因此，应尽量减少缺失:
  - 多路组相连
  - 再尽量提高TLB的容量



转换旁路缓冲（TLB）

- TLB 可以组织成全相连，组相连或直接映射方式
- TLBs 通常为容量较小，甚至在高端计算机上也一般不超过128 – 256 个表项。这样，可以使用全相连映射方式。在大多数中档计算机上，一般采用N路组相联映射方式。



Translation with a TLB

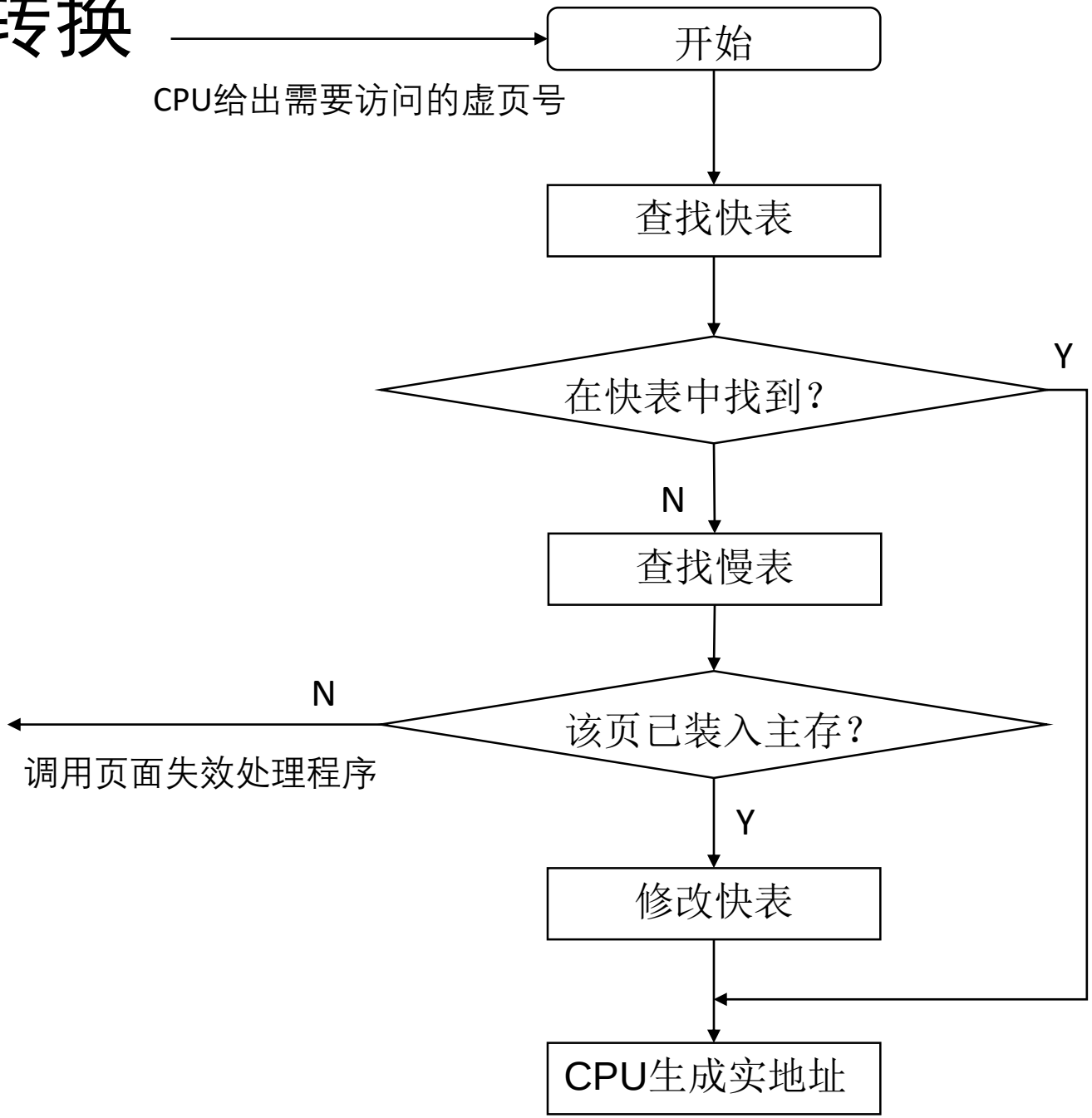
页面大小的选择

- 减少内部碎片
  - 缩小页面大小可以减少内部碎片
  - 但是：需要更大的页表
- 趋势：增大页面大小
  - RAM价格下降，内存储器容量增大
  - 内存和外存性能差距增大
  - 程序员需要更大的地址空间
- 目前：页面大小为4K左右？（1MB, 2MB, 4MB, 1GB)
  - Linux内核的Huge Page机制

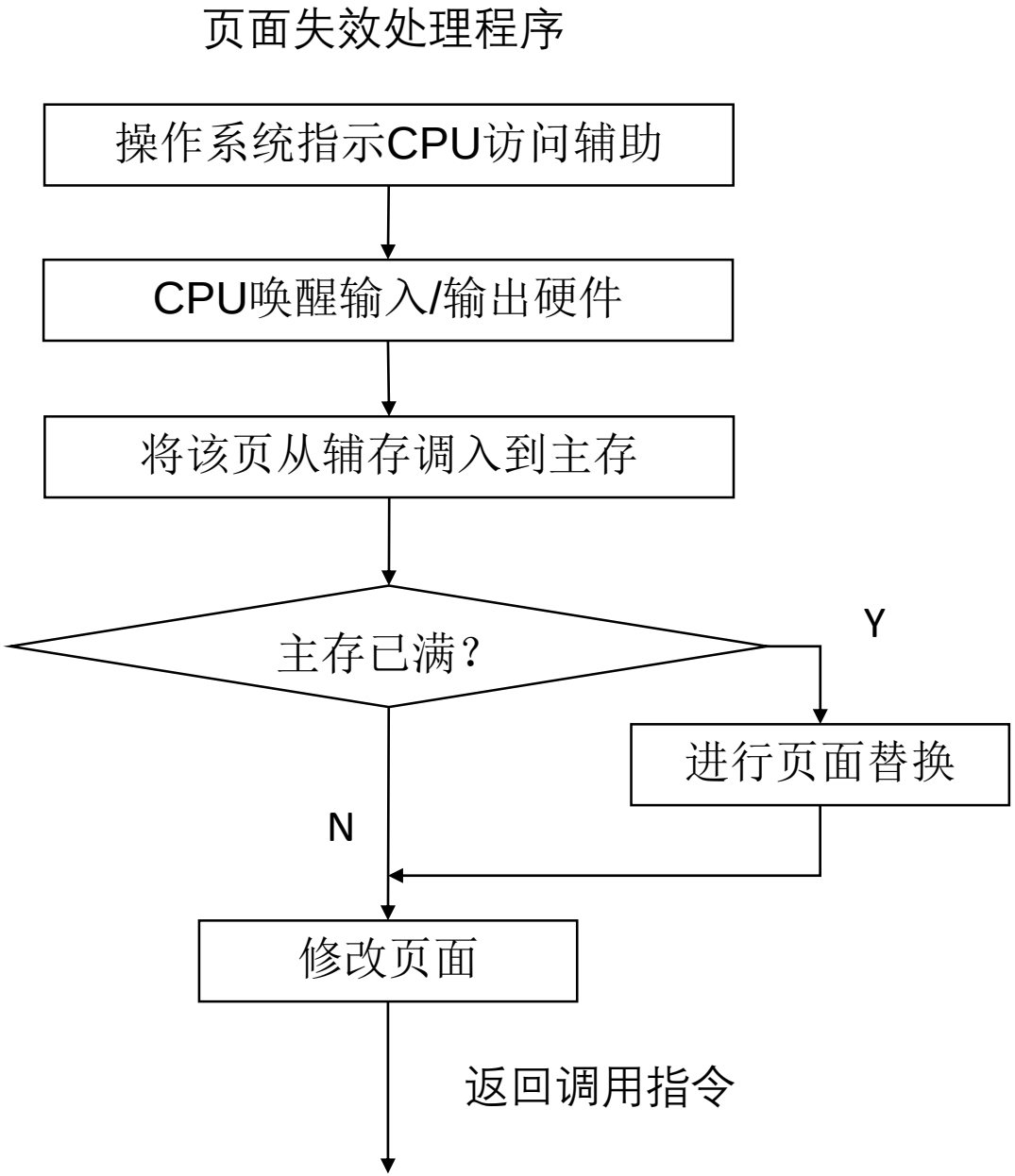
页面替换算法

- 最近最少使用（LRU）
  - 将页帧按照最近最多使用到最近最少使用进行排序，再次访问一个页帧时，将该页帧移到表头，替换时将表尾的页帧换出。
  - 一点改进：替换出其中一个“干净”的页帧。

地址转换


开始
CPU给出需要访问的虚页号
查找快表
在快表中找到？
Y
N
查找慢表
该页已装入主存？
N
调用页面失效处理程序
Y
修改快表
CPU生成实地址

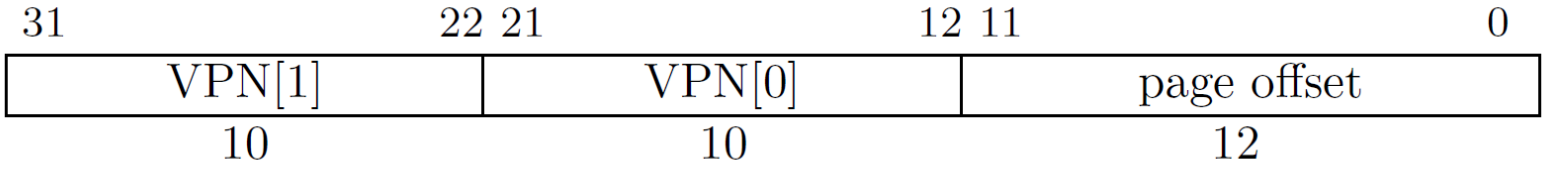
页失效处理


页面失效处理程序
操作系统指示CPU访问辅助
CPU唤醒输入/输出硬件
将该页从辅存调入到主存
主存已满？
Y
进行页面替换
N
修改页面
返回调用指令

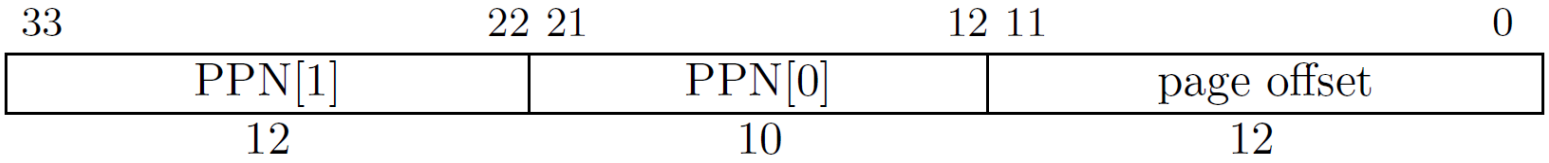
基于页面的虚拟内存

- 分页命名模式：SvX，其中X是以位为单位的虚拟地址长度
- 内存划分为固定大小的页面进行地址转换和对内存内容的保护（页面大小通常为4KB，也有大页面粒度）
- 启用分页的时候，大多数地址（包括load和store的有效地址和PC中的地址）都是虚拟地址
- 要访问物理内存，虚拟地址必须被转换为真正的物理地址
- 通过页表的结构来进行转换
- 权限位指示那些权限模式和通过哪种类型的访问可以操作这个页
- 访问未被映射的页或者访问权限不足会导致页面错误异常（page fault exception）

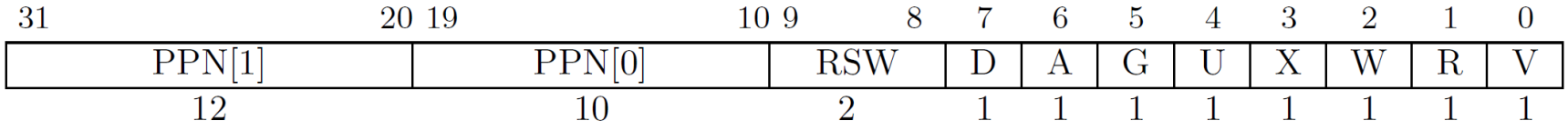
Sv32的虚拟地址与物理地址



Sv32虚拟地址

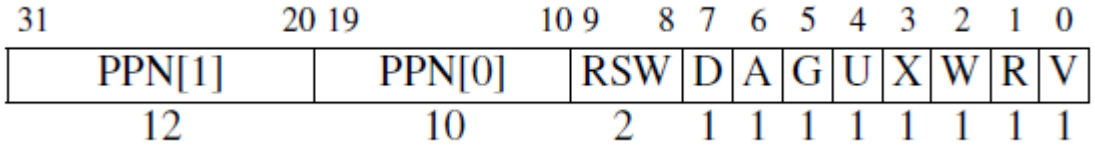


Sv32物理地址



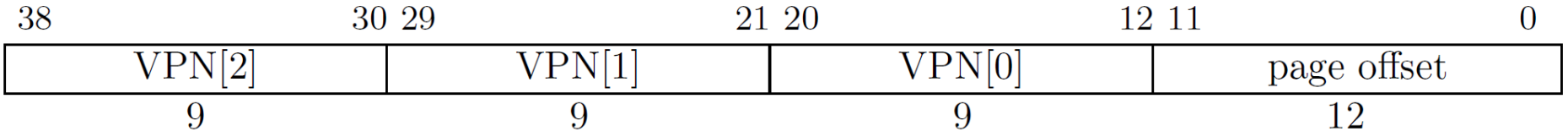
Sv32页表项

RV32 Sv32页表项（PTE: page table entry）

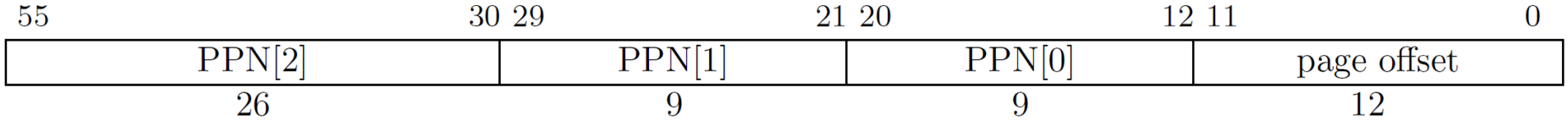


- V：有效位
- R,W,X：读，写，执行位
- U：0代表U模式不能访问，但是S模式可以；1代表U模式可以访问，S模式可以
- G：Global是否对所有地址空间有效
- A：Access，是否被访问过
- D：Dirty，是否被修改过
- RSW：操作系统使用，被硬件忽略
- PPN：物理页号，这是物理地址的一部分。如果这个页表项是一个叶子结点，那么PPN是转换后物理地址的一部分。否则PPN给出的是下一级页表的地址

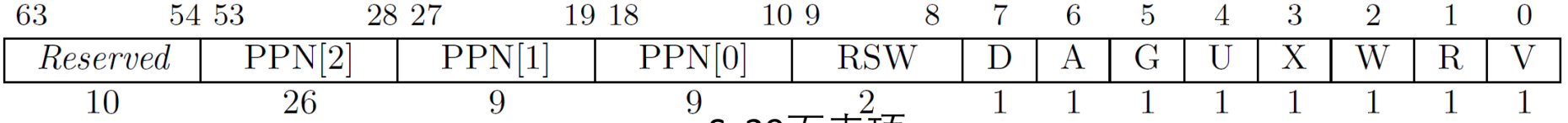
Sv39的虚拟地址与物理地址



Sv39虚拟地址



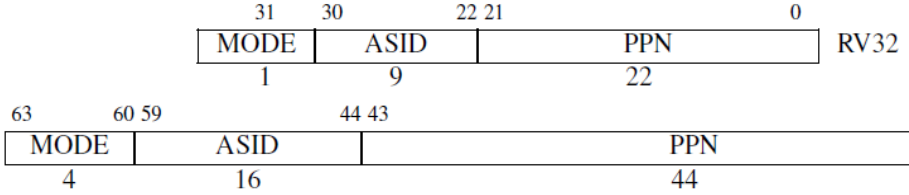
Sv39物理地址



Sv39页表项

satp寄存器

- satp（Supervisor Address Translation and Protection，监管者地址转换和保护） S 模式控制状态寄存器控制了分页系统
- ASID：Address Space Identifier，地址空间标识符（可选），用以降低上下文切换开销



PPN指向根页表的物理地址

RV32		
Value	Name	Description
0	Bare	No translation or protection.
1	Sv32	Page-based 32-bit virtual addressing.

RV64		
Value	Name	Description
0	Bare	No translation or protection.
8	Sv39	Page-based 39-bit virtual addressing.
9	Sv48	Page-based 48-bit virtual addressing.

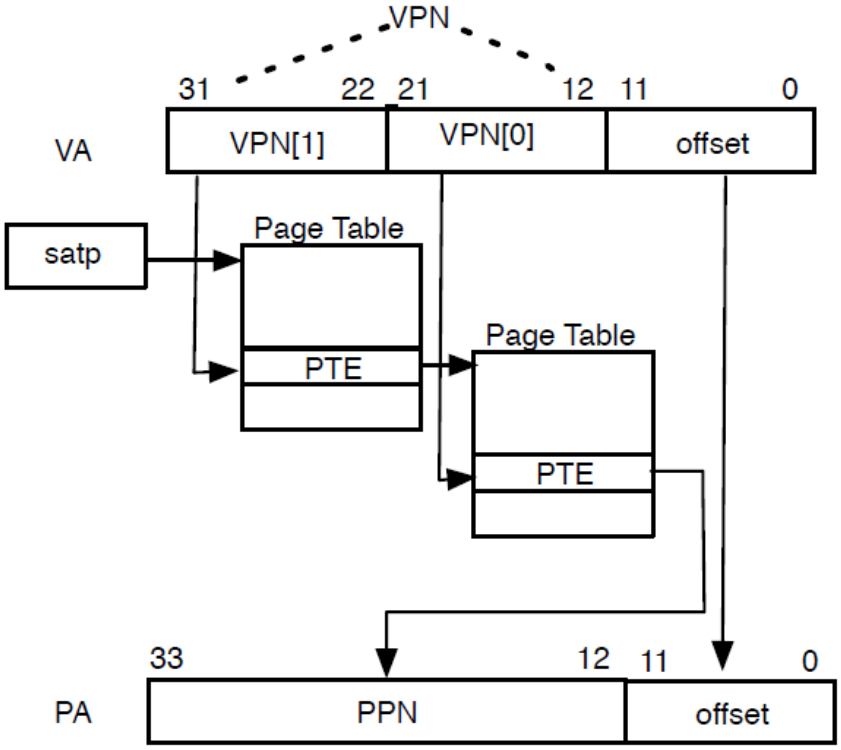
MODE域编码

satp寄存器初始化

- M模式的程序在第一次进入S模式之前会把0写入satp，以禁用分页
- 然后S模式的程序在初始化页表以后会再次进行satp寄存器的写操作

虚拟地址到物理地址的转换

- satp.PPN 给出了一级页表的基址，VA[31:22]给出了一级页号，因此处理器会读取位于地址(satp.PPN × 4096 + VA[31: 22] × 4)的页表项。
- 该 PTE 包含二级页表的基址，VA[21:12]给出了二级页号，因此处理器读取位于地址(PTE.PPN × 4096 + VA[21: 12] × 4)的叶节点页表项。
- 叶节点页表项的 PPN 字段和页内偏移（原始虚址的最低 12 个有效位）组成了最终结果：物理地址就是(LeafPTE. PPN × 4096 + VA[11: 0])



虚拟地址到物理地址的转换

1. Let a be `satp`.$ppn \times$ PAGESIZE, and let $i =$ LEVELS $- 1$.
2. Let pte be the value of the PTE at address $a + va.vpn[i] \times$ PTESIZE.
3. If $pte.v = 0$, or if $pte.r = 0$ and $pte.w = 1$, stop and raise a page-fault exception.
4. Otherwise, the PTE is valid. If $pte.r = 1$ or $pte.x = 1$, go to step 5. Otherwise, this PTE is a pointer to the next level of the page table. Let $i = i - 1$. If $i < 0$, stop and raise a page-fault exception. Otherwise, let $a = pte.ppn \times$ PAGESIZE and go to step 2.
5. A leaf PTE has been found. Determine if the requested memory access is allowed by the $pte.r$, $pte.w$, $pte.x$, and $pte.u$ bits, given the current privilege mode and the value of the SUM and MXR fields of the `mstatus` register. If not, stop and raise a page-fault exception.
6. If $i > 0$ and $pa.ppn[i - 1 : 0] \neq 0$, this is a misaligned superpage; stop and raise a page-fault exception.
7. If $pte.a = 0$, or if the memory access is a store and $pte.d = 0$, then either:
   - Raise a page-fault exception, or:
   - Set $pte.a$ to 1 and, if the memory access is a store, also set $pte.d$ to 1.
8. The translation is successful. The translated physical address is given as follows:
   - $pa.pgoff = va.pgoff$.
   - If $i > 0$, then this is a superpage translation and $pa.ppn[i - 1 : 0] = va.vpn[i - 1 : 0]$.
   - $pa.ppn[\text{LEVELS} - 1 : i] = pte.ppn[\text{LEVELS} - 1 : i]$.

虚址到物理地址转换的完整算法。va 是输入的虚拟地址，pa 是输出的物理地址。PAGESIZE 是常数 2^{12}。在 Sv32 中，LEVELS = 2 且 PTESIZE = 4；而在 Sv39 中，LEVELS = 3 且PTESIZE = 8

TLB

- 如果取指，load，store要访问多次页表，将会大大降低性能
- 所有的现代处理器都使用地址转换缓存（TLB：Translation Lookaside Buffer）来减少这种开销
- 如果操作系统修改了页表，TLB就会变得不可用
- sfence.vma通知处理器，软件可能已经修改了页表，处理器可以刷新TLB
  - rs1指示哪个虚拟地址对应的转换被修改了
  - rs2指示被修改页表的地址空间标识符（一般相当于进程）ASID
  - 如果两者都是x0，整个TLB会被刷新

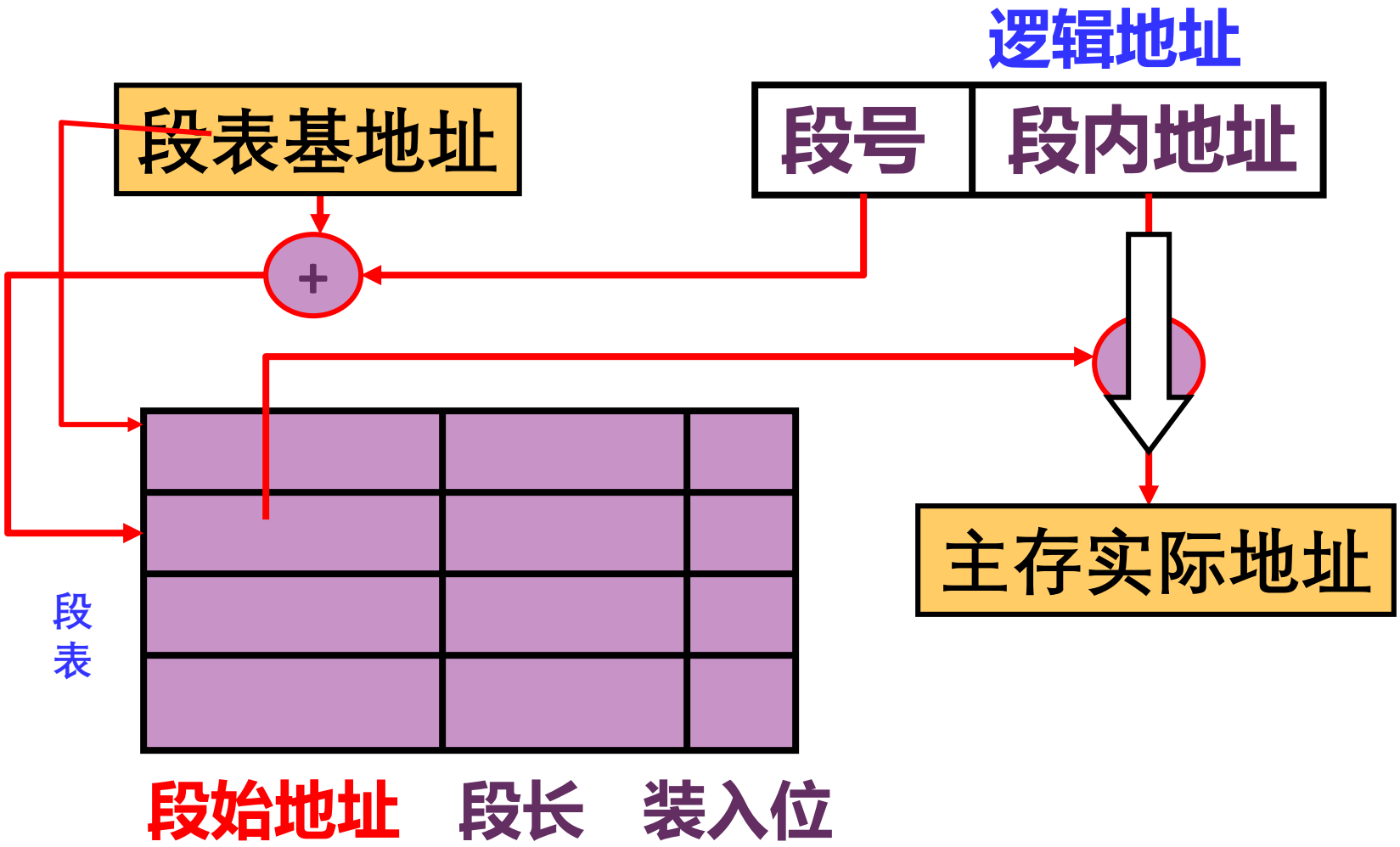
内容提要

- 为什么需要虚拟内存？
- 页式内存管理
  - TLB
- 例子：RISC-V 页表管理
- 段式内存管理与段页式内存管理
- 例子：x86 页表管理

段式存储管理的实现

- 设置段表进行管理
  - 段表基地址
  - 段起始地址
  - 段长
  - 装入位
  - 保护、共享等标志

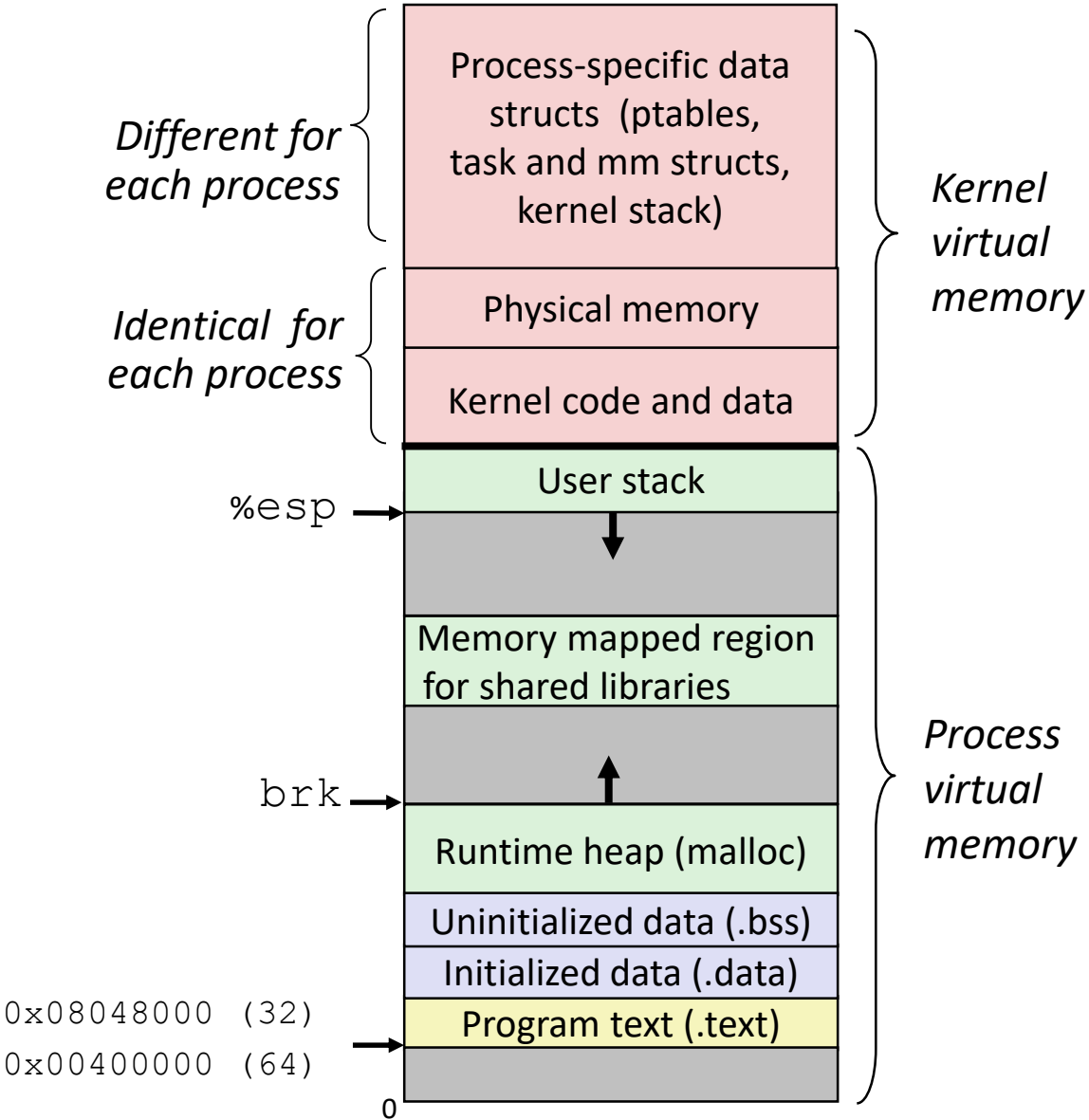
段式管理地址转换



段式存储管理

- 段的分界与程序和数据的自然分界相对应
- 易于编译、管理、修改和保护，便于多道程序共享
- 段长动态可变（？）
- 段起点、终点不定（？）
- 空间分配困难，容易产生碎片

Virtual Memory of a Linux Process



段页式虚拟存储管理

- 是段式虚拟存储器和页式虚拟存储器的综合。它先把程序按逻辑单位分为段，再把每段分成固定大小的页。操作系统对主存的调入调出是按页面进行的，但它又可以按段实现共享和保护，可以兼取页式和段式系统的优点。其缺点是需要在地址映射过程中多次查表。其地址映射通过一个段表和一组页表来进行。

x86的虚存管理

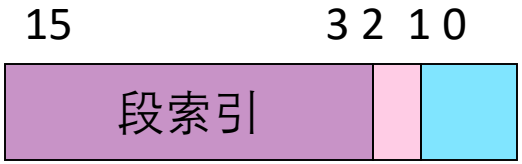
- 不分段也不分页模式：在这种模式下，虚拟存储的地址空间和物理存储空间大小相同，可以用在复杂度较低但对性能有较高要求的场合。
- 页式管理模式：这种模式将主存分成固定长度的页，通过页进行存储保护和管理。
- 段式管理模式：段式管理模式按程序本身的逻辑段来划分主存空间，与页式管理相比，段的长度可变
- 段页式管理模式：为了兼容旧的模式，在x86中段式内存管理和页式内存管理都是支持的。程序地址首先经过段式内存转换，再通过页式内存转换，最终转换为物理地址。

32位x86虚实地址的转换

- 虚地址（逻辑地址）：
  - 程序员给出的虚拟地址，格式为段号+段内偏移（16位+32位），每段大小不超过4GB，一共不超过2^{14}段。（段号中有两位用来表示段优先级）
- 实地址：
  - 32位的实际内存地址。

段号和段表的格式

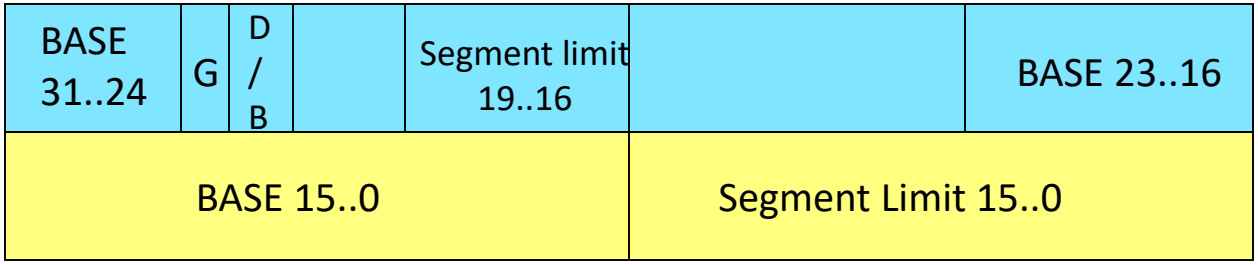
段号:



0: GDT 1: LDT

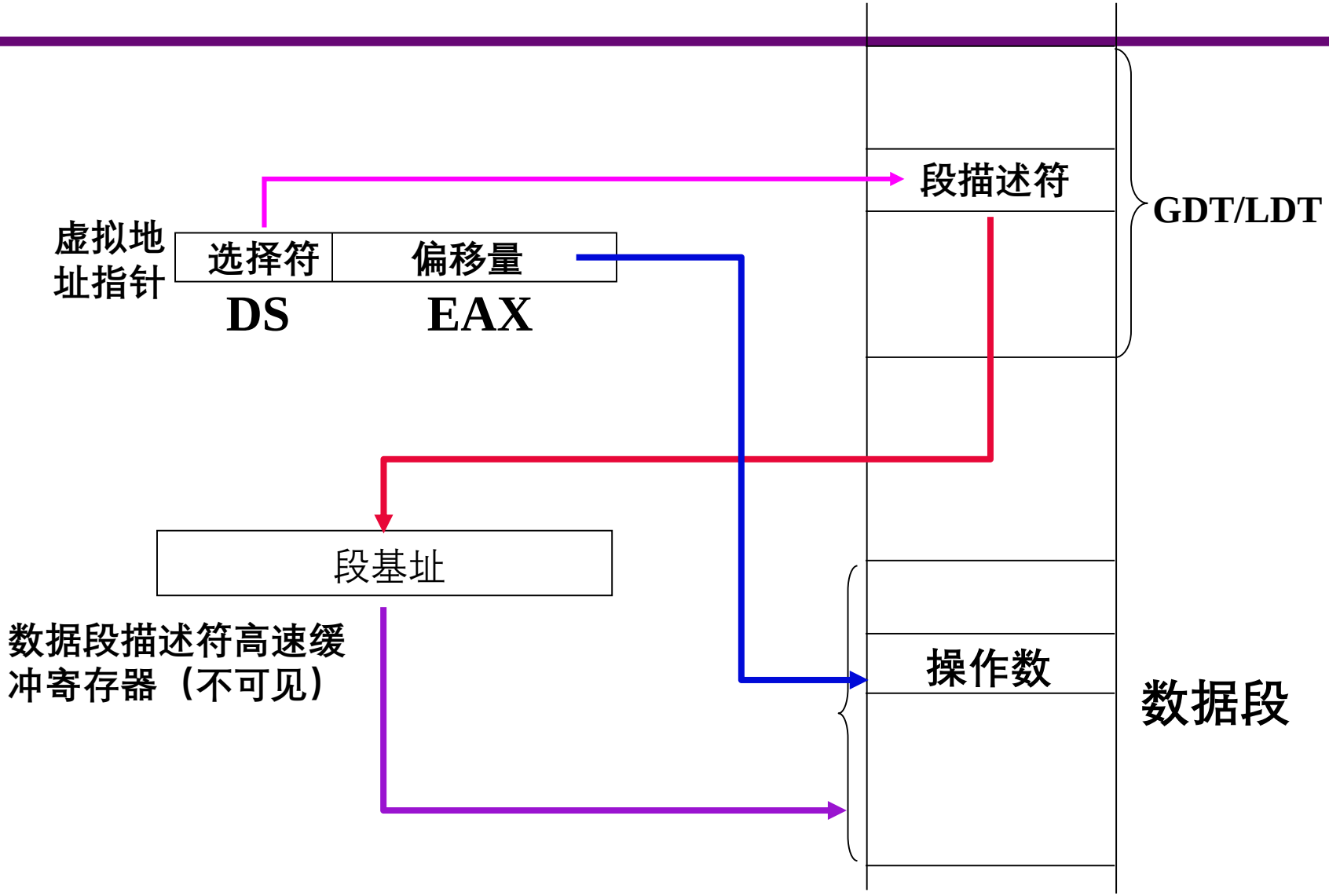
段表基地址

段表:

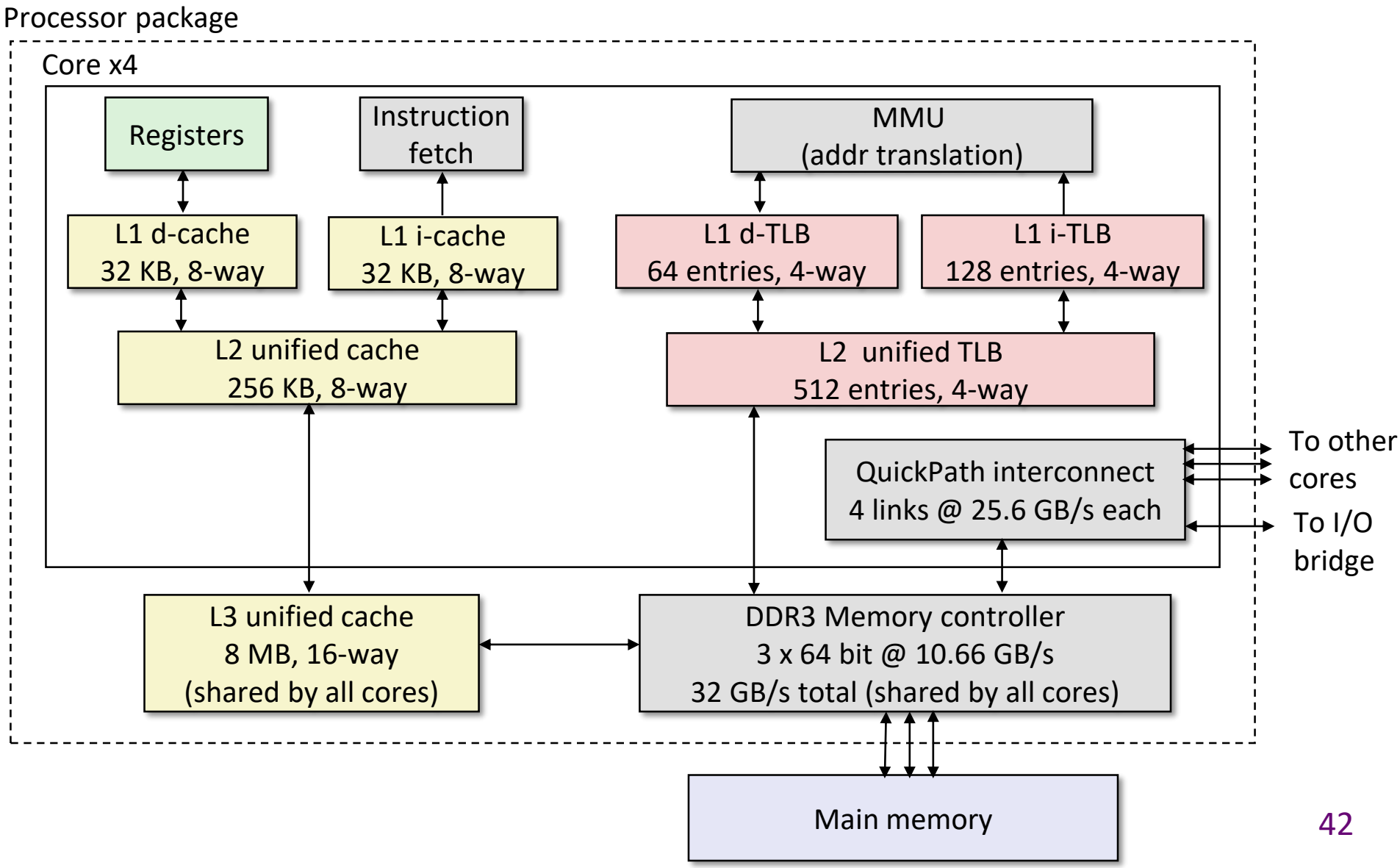


线性地址=BASE + OFFSET

段地址转换


虚拟地址指针
选择符
偏移量
DS
EAX
段描述符
GDT/LDT
段基址
数据段描述符高速缓冲寄存器（不可见）
操作数
数据段

Intel Core i7 Memory System

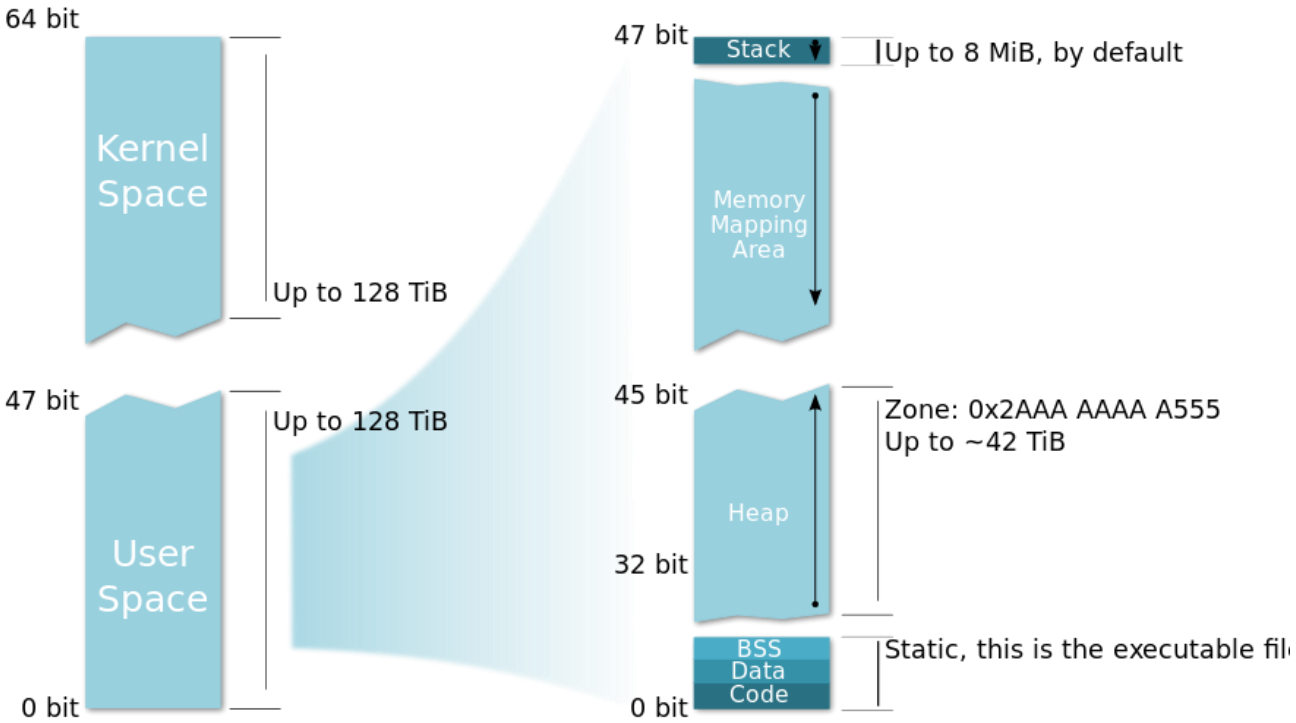


Review of Symbols

- Basic Parameters
  - $N = 2^n$: Number of addresses in virtual address space
  - $M = 2^m$: Number of addresses in physical address space
  - $P = 2^p$: Page size (bytes)
- Components of the virtual address (VA)
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: Virtual page offset
  - VPN: Virtual page number
- Components of the physical address (PA)
  - PPO: Physical page offset (same as VPO)
  - PPN: Physical page number
  - CO: Byte offset within cache line
  - CI: Cache index
  - CT: Cache tag

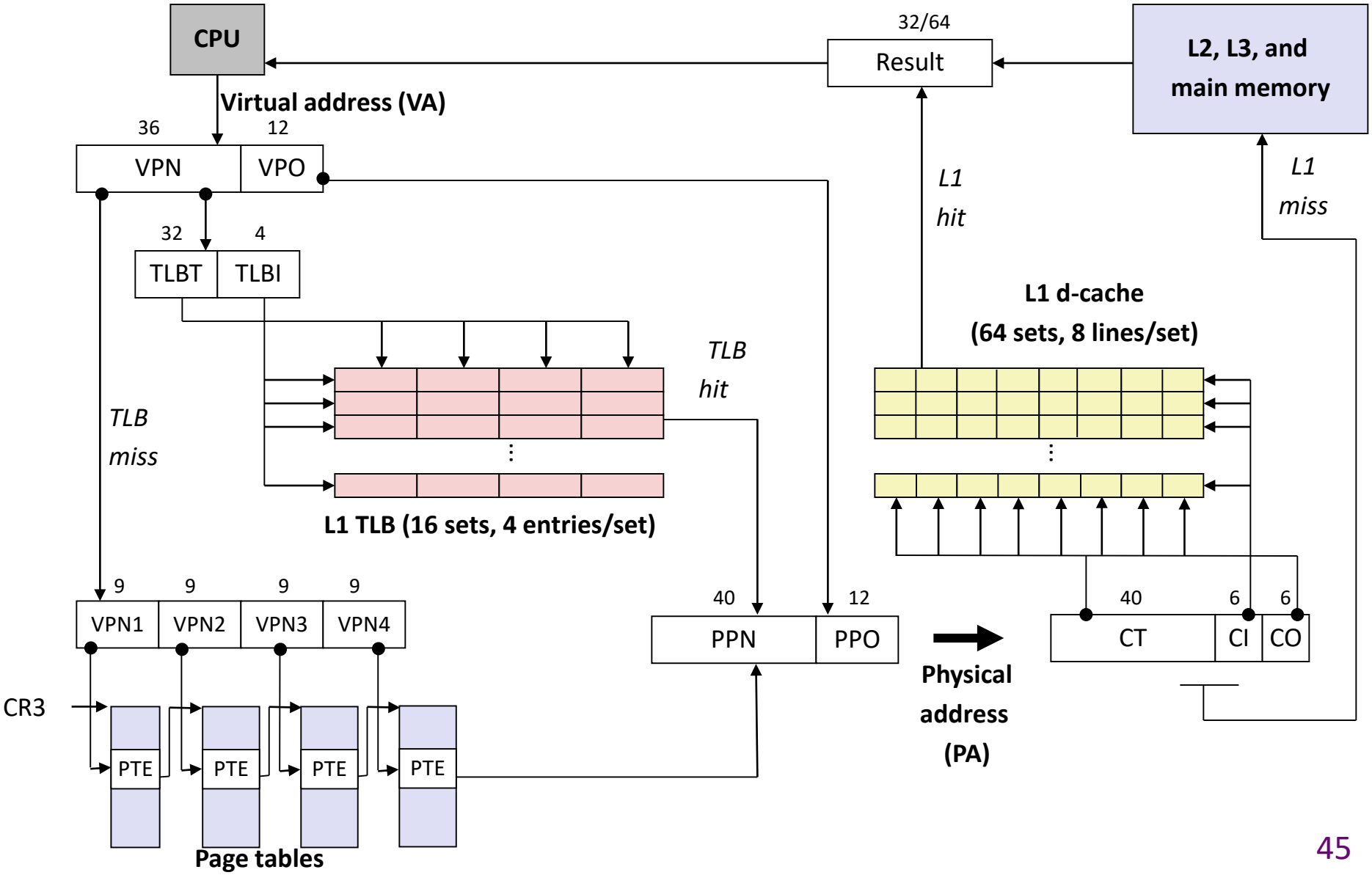
X86-64 Linux layout

- **Only 48 of 64bit used for virtual memory**

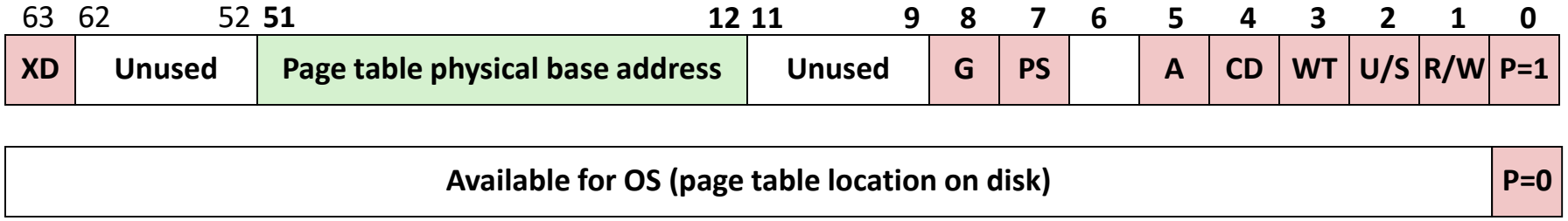


So Kernel + User Spaces add for 256 TiB which is a tiny part of the 16 777 216 TiB addressable over 64 bit!

End-to-end Core i7 Address Translation



Core i7 Level 1-3 Page Table Entries



Each entry references a 4K child page table

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

CD: Caching disabled or enabled for the child page table.

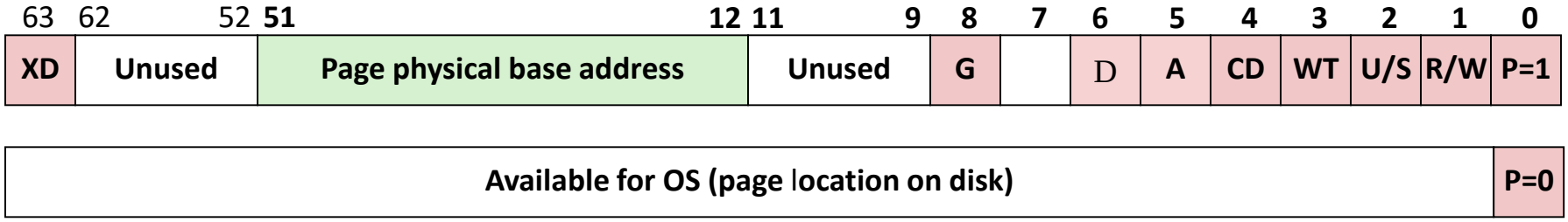
A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).

G: Global page (don't evict from TLB on task switch)

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

Core i7 Level 4 Page Table Entries



Each entry references a 4K child page

P: Child page is present in memory (1) or not (0)

R/W: Read-only or read-write access permission for child page

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

CD: Cache disabled (1) or enabled (0)

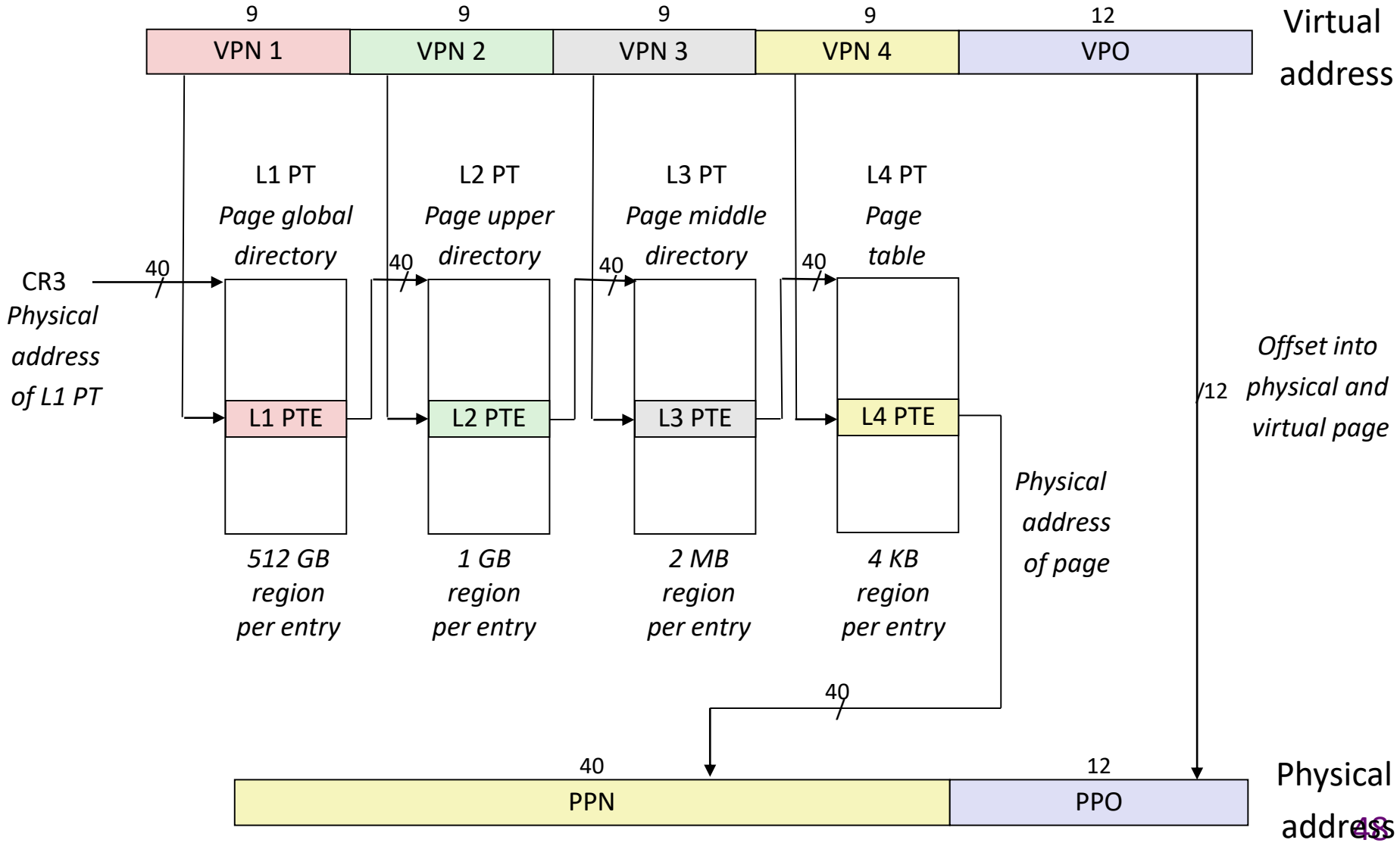
A: Reference bit (set by MMU on reads and writes, cleared by software)

D: Dirty bit (set by MMU on writes, cleared by software)

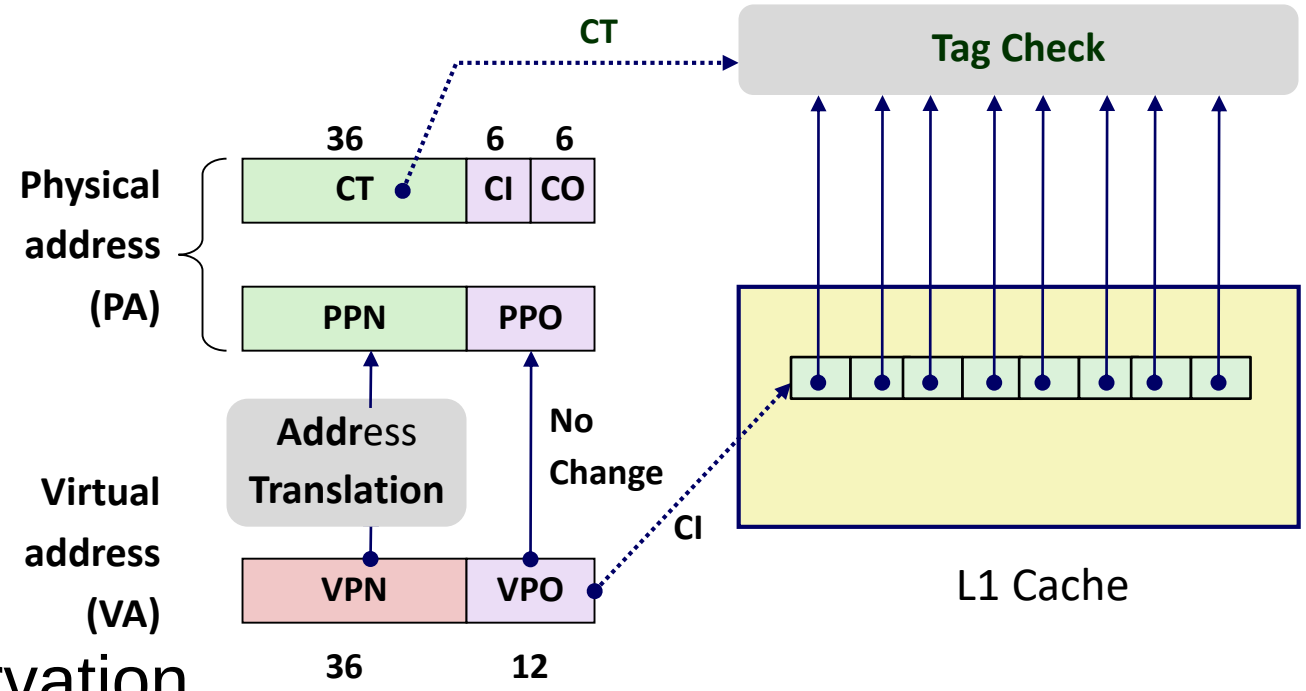
G: Global page (don't evict from TLB on task switch)

Page physical base address: 40 most significant bits of physical page address (forces pages to be 4KB aligned)

Core i7 Page Table Translation



Cute Trick for Speeding Up L1 Access

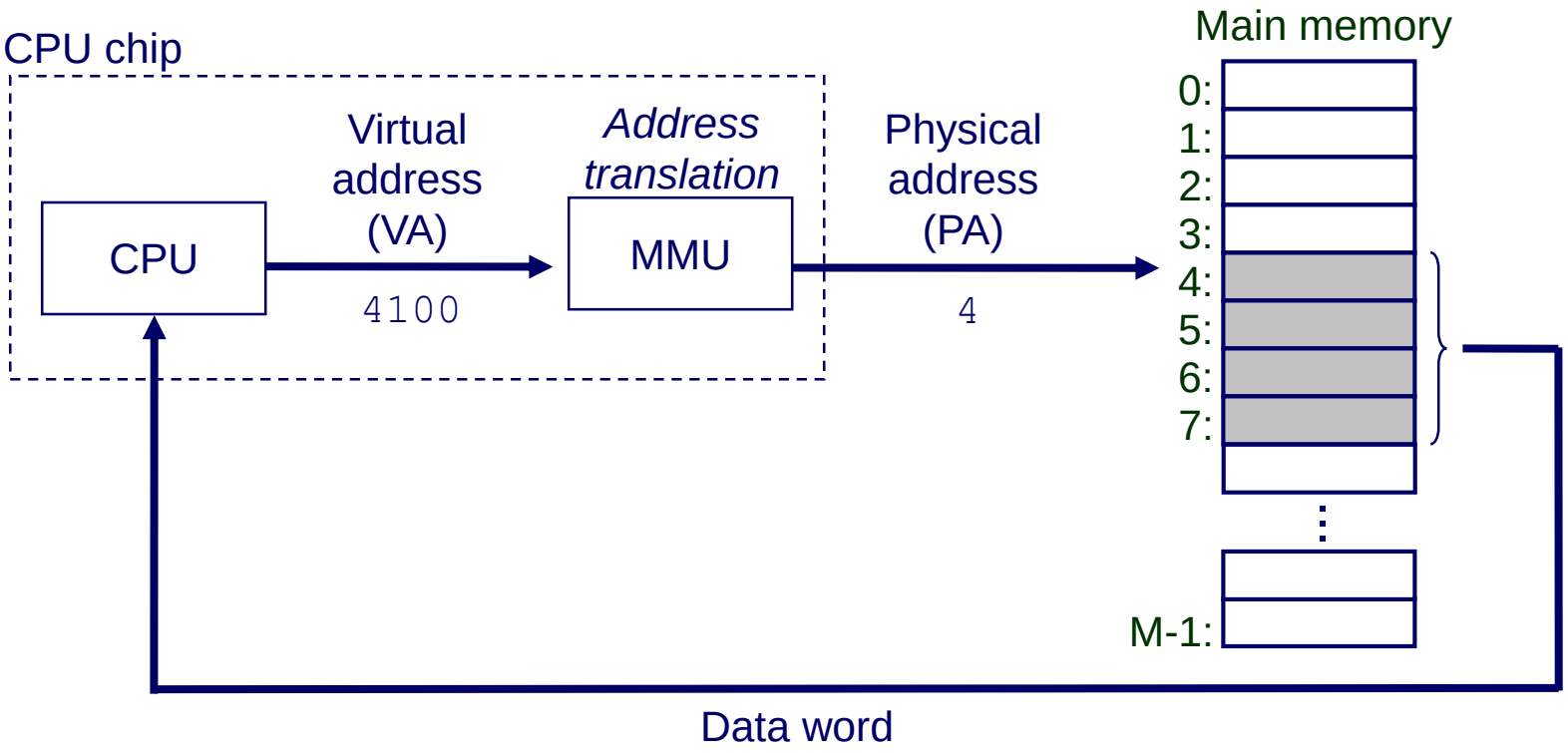


- Observation
  - Bits that determine CI identical in virtual and physical address
  - Can index into cache while address translation taking place
  - Generally we hit in TLB, so PPN bits (CT bits) available next
  - “Virtually indexed, physically tagged”
  - Cache carefully sized to make this possible

阅读和思考

- 阅读
  - 教材相关章节
- 思考
  - 随着主存容量的扩大，是否还有继续使用虚拟存储器的必要？
- 实践
  - 继续完成实验报告

谢谢


CPU chip
CPU
Virtual address (VA)
4100
Address translation
MMU
Physical address (PA)
4
Main memory
0:
1:
2:
3:
4:
5:
6:
7:
M-1:
Data word