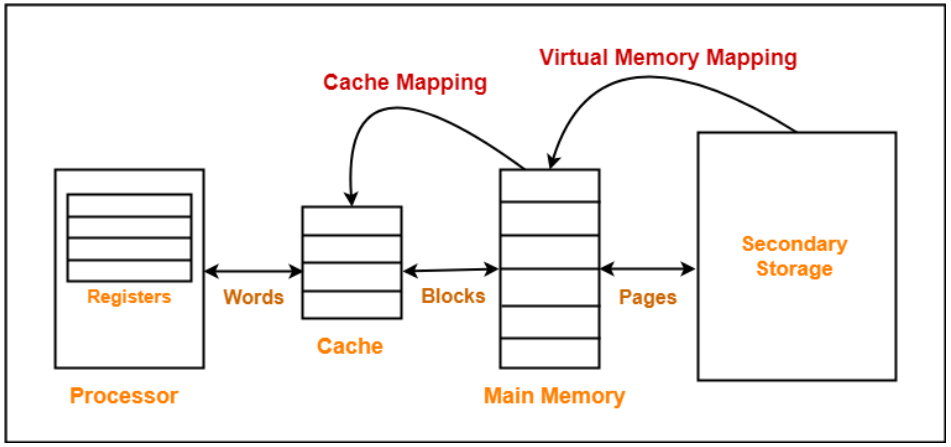
计算机组成原理

RV $5.7 虚拟存储器

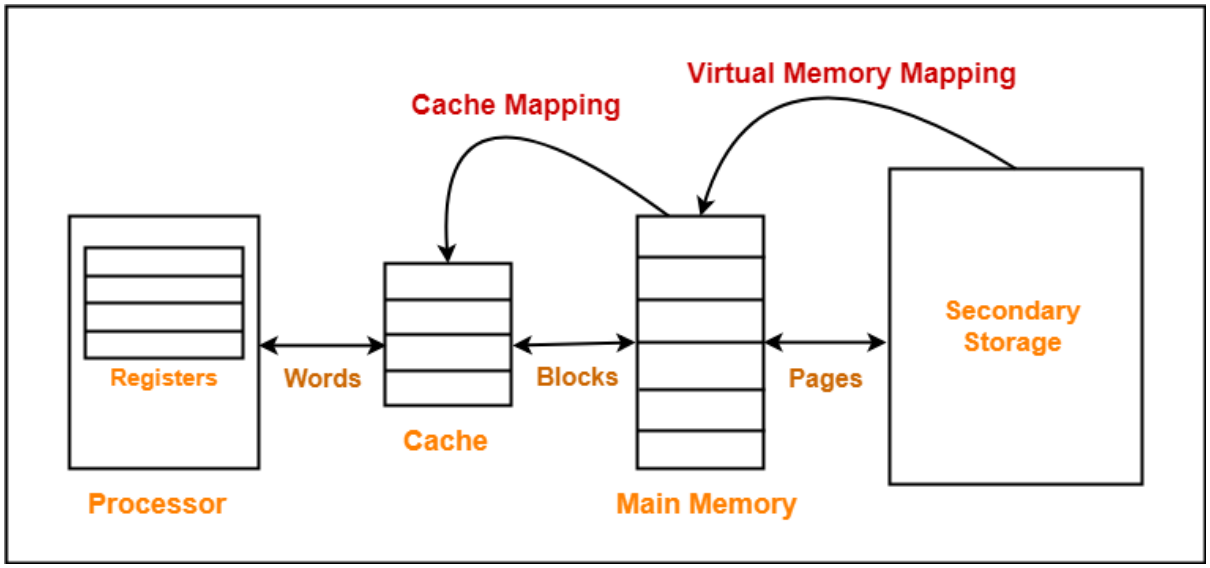
llxx@ustc.edu.cn

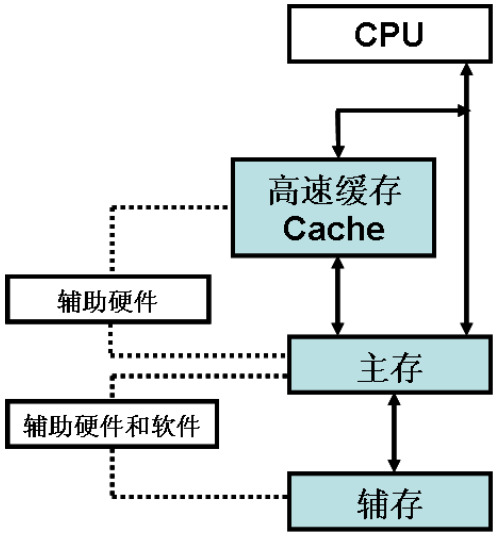
本章内容：COD5 $5.7节

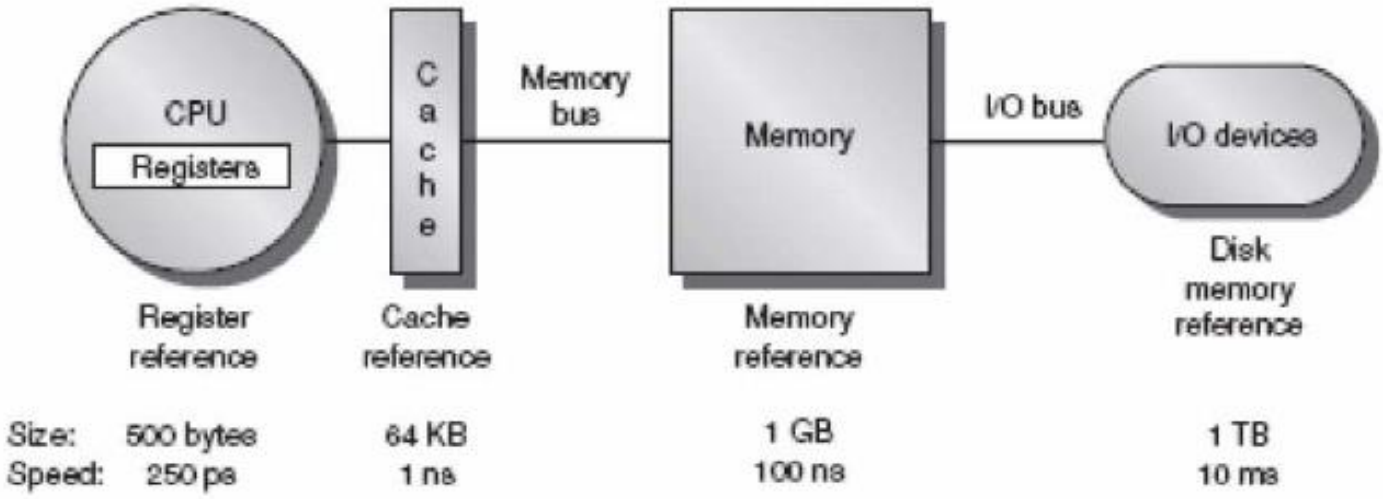
- ✓实地址 vs 虚地址
- ✓虚存技术动机
- ✓页式虚存管理原理
  - ✓PWR
- ✓页式虚存管理设计：实例
  - ✓TLB
  - ✓MMU
- ✓层次化访存过程
  - ✓Cache-TLB-Memory-Disk
  - ✓缺页异常处理



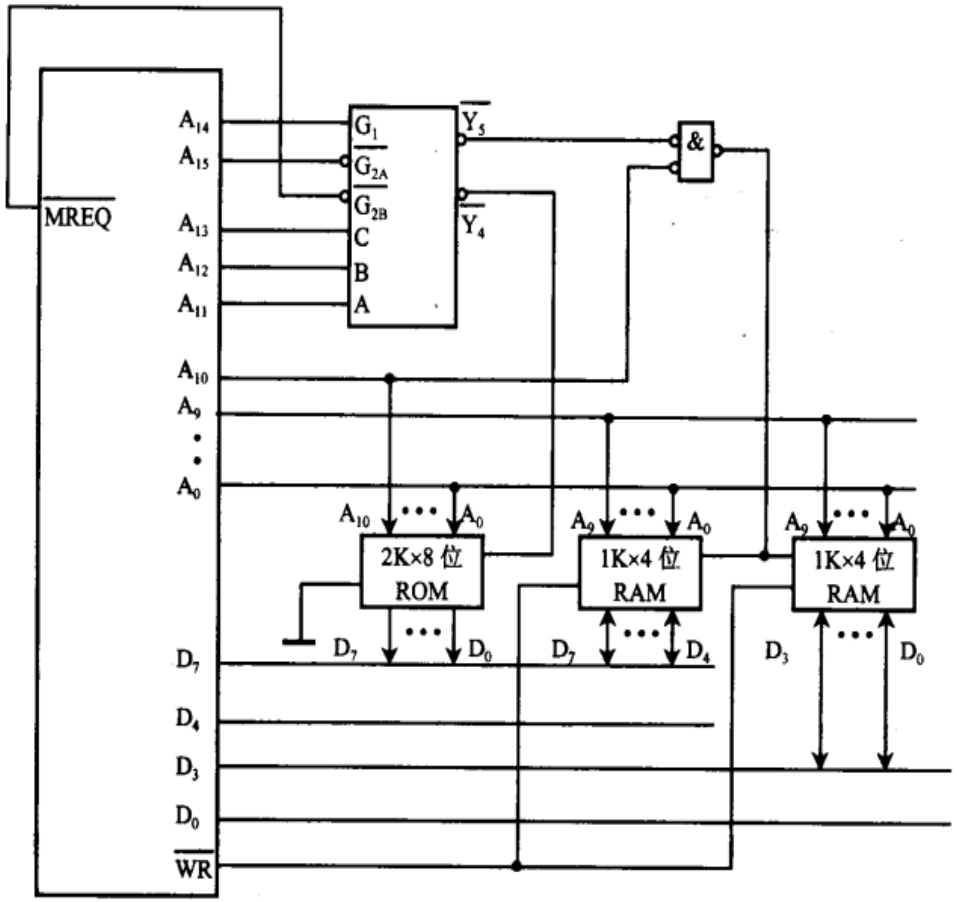
层次化存储系统：访存时间?

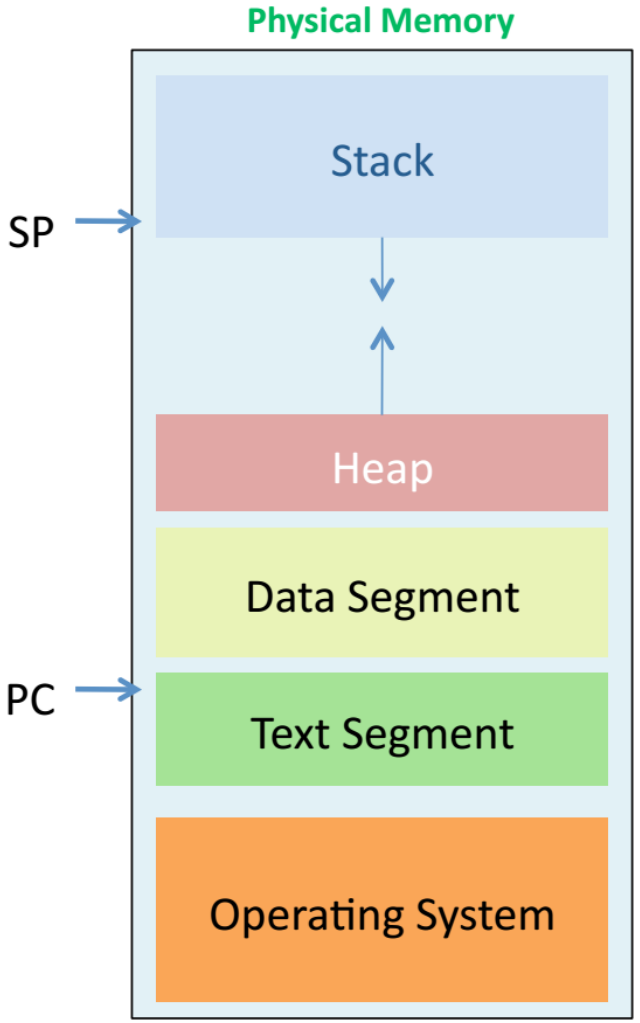






实地址访存：存储器物理地址

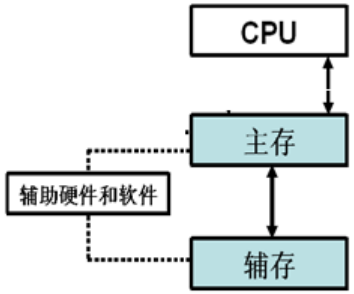




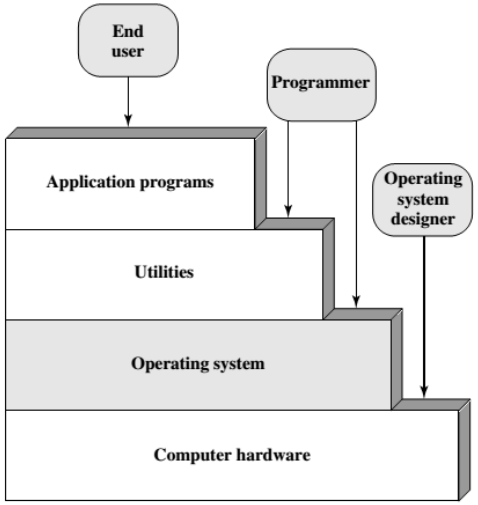
不利于多任务（须预先划分每个程序占用的内存范围）

段式

虚拟存储器（Virtual memory）

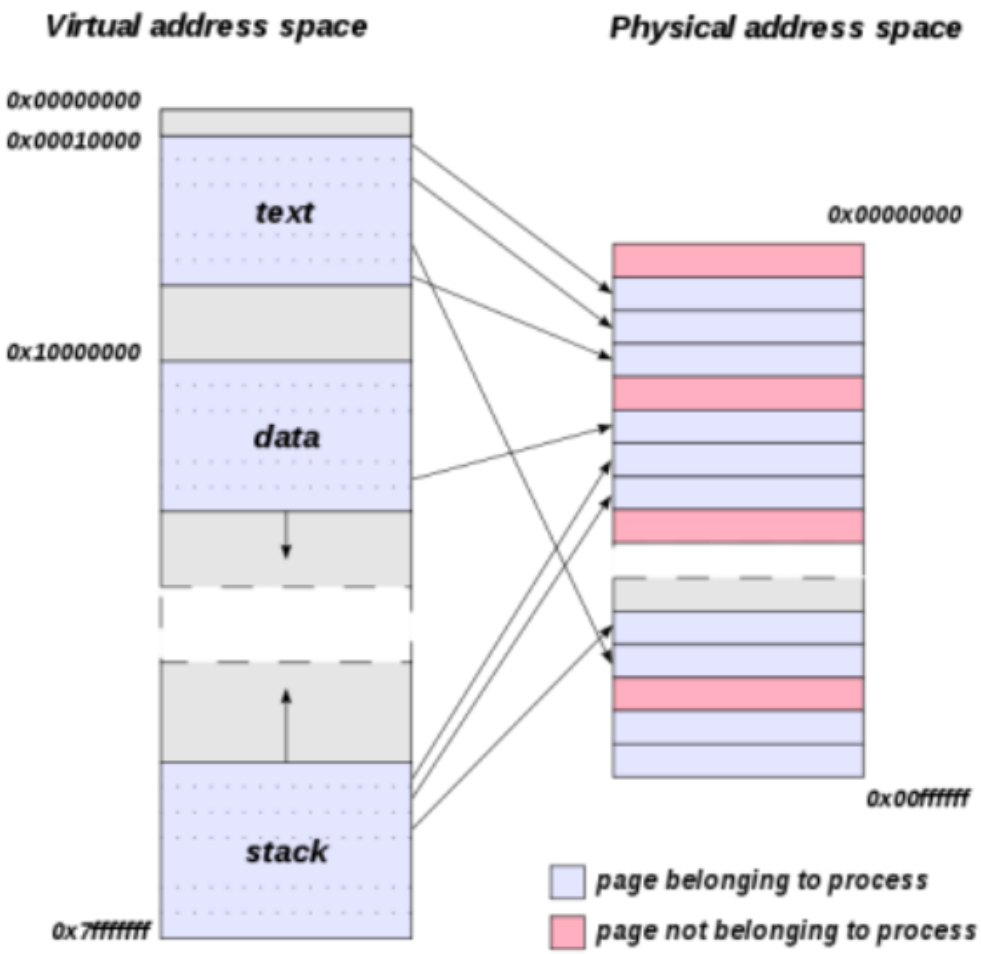


- 早期：1961年曼彻斯特大学提出
  - 内存容量：程序要求的存储器空间越来越大
    - 模式一：虚存= 主存 + 辅存
    - Overlay技术：程序分段，段长 < 内存大小；程序员负责换入换出
  - 多道程序：代码和数据保护与共享存储
    - 模式二：主存作为辅存（**虚存**）的Cache
- 现代虚拟存储系统
  - 多用户多进程
    - 性能：一种将主存作为辅存的缓存的技术
    - 模式二，虚存驻留于辅存，局部性原理

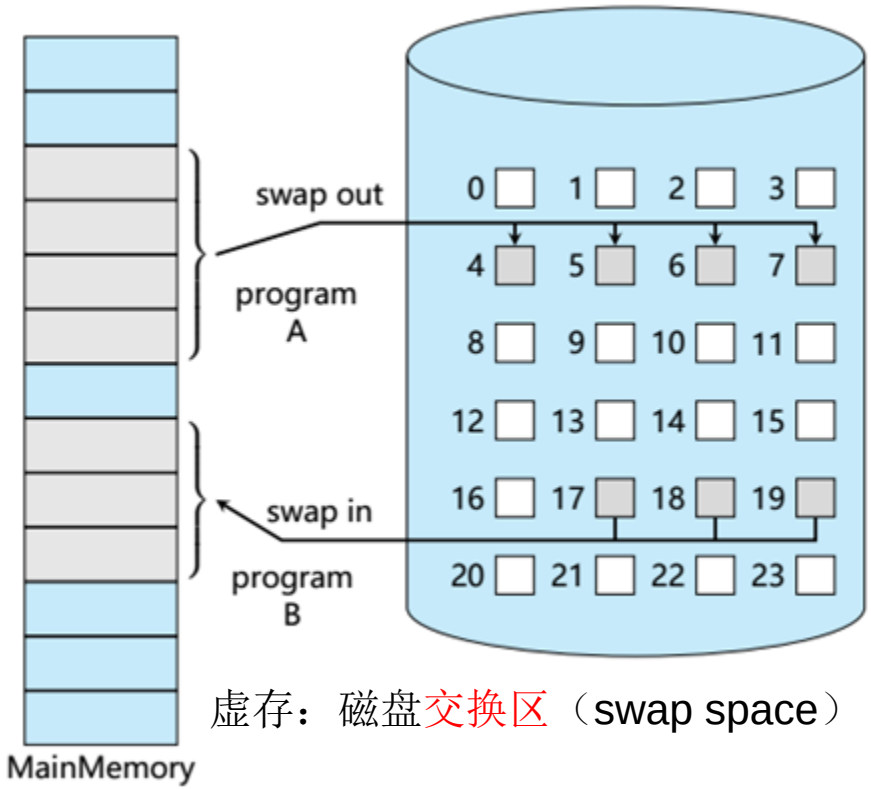


  - 由MMU和OS存储管理器共同管理：对普通程序员透明
    - 虚方式访存：重定位（relocation）技术，将虚地址映射到物理地址
    - 页式（定长），段式（可变长），段页式

程序的地址空间：虚地址空间



逻辑空间 <=> 物理空间 <==> 磁盘空间



虚存：磁盘交换区（swap space）

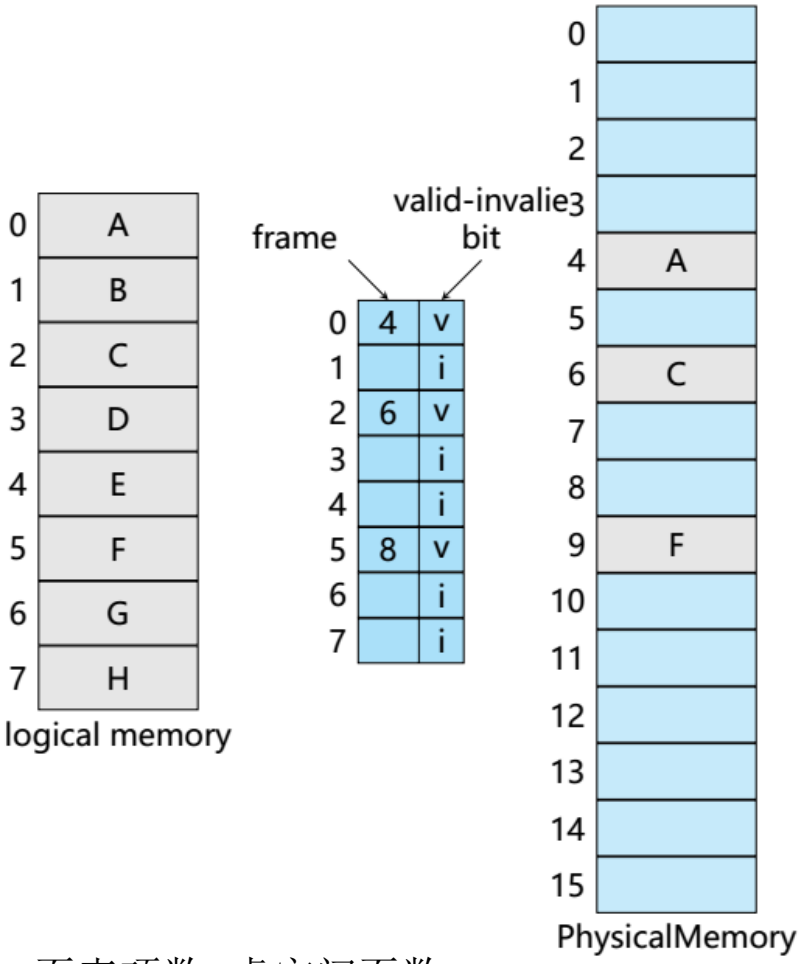
换入/换出时机？

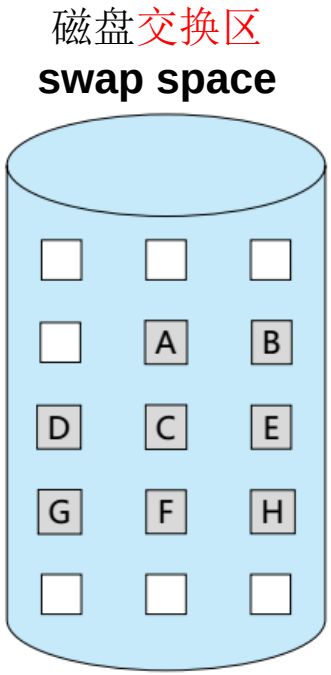
Demand Paging策略

Bring a page into memory only when it is needed

1. Less I/O needed
2. Less memory needed
3. Faster response
4. More users

主存作为虚存的Cache：PWR





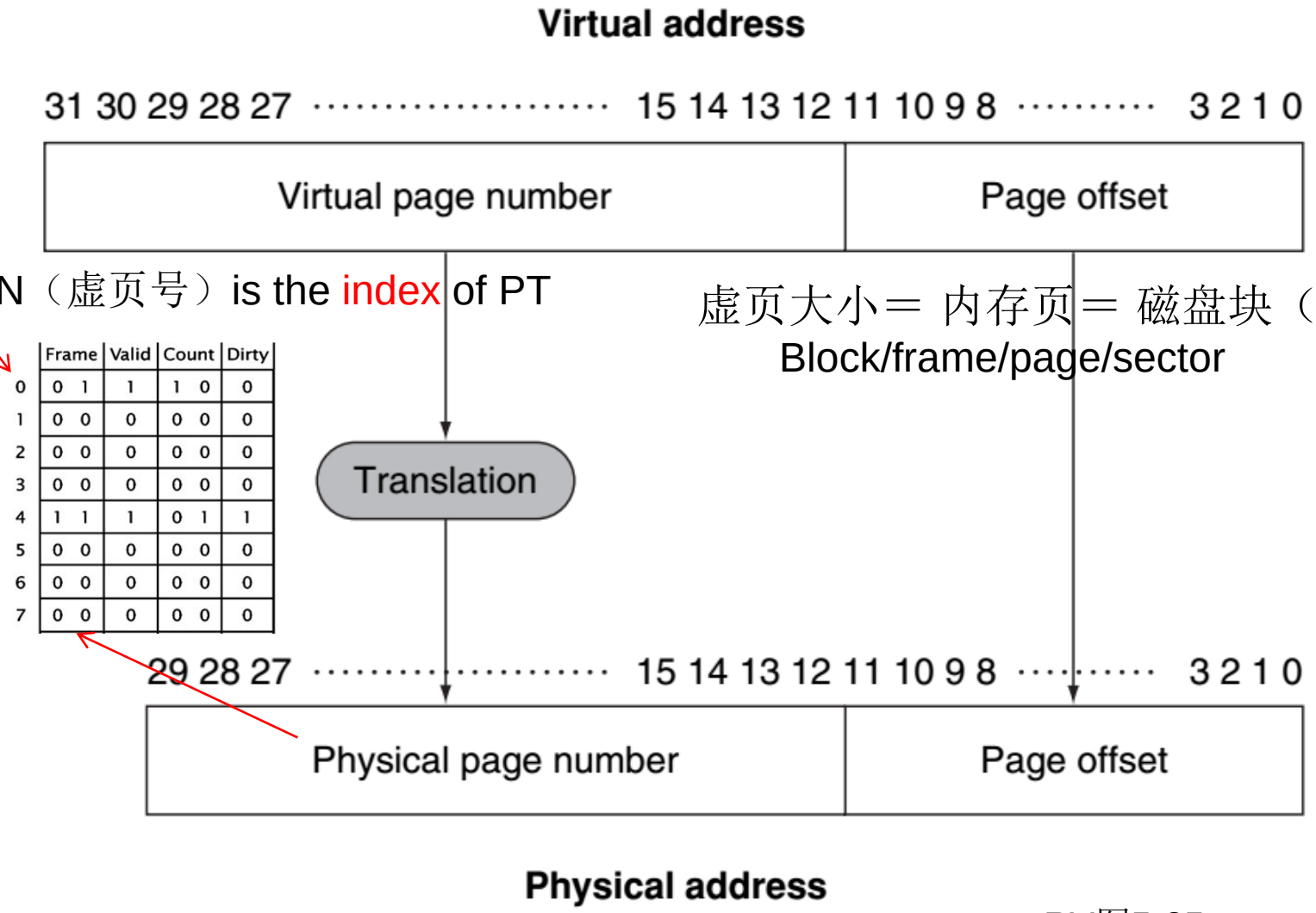
虚存：在辅存中

分页：虚实页大小相等。
全相联：虚页可在内存任意位置
　虚实映射：按页表查找
　页表：在内存中。大小？
写操作：写回
　When？一致性？
替换

OS在创建进程时，为每个进程建立其页表和磁盘上的VA空间。
由OS+MMU管理，对程序员和编译器透明！

页表项数=虚空间页数
页状态：Valid-Invalid(PRESENT，pagefault)，reference（替换），dirty

虚实地址转换：虚实映射，页表



- VPN（虚页号）is the index of PT

虚页大小＝ 内存页＝ 磁盘块（n 扇段）
Block/frame/page/sector

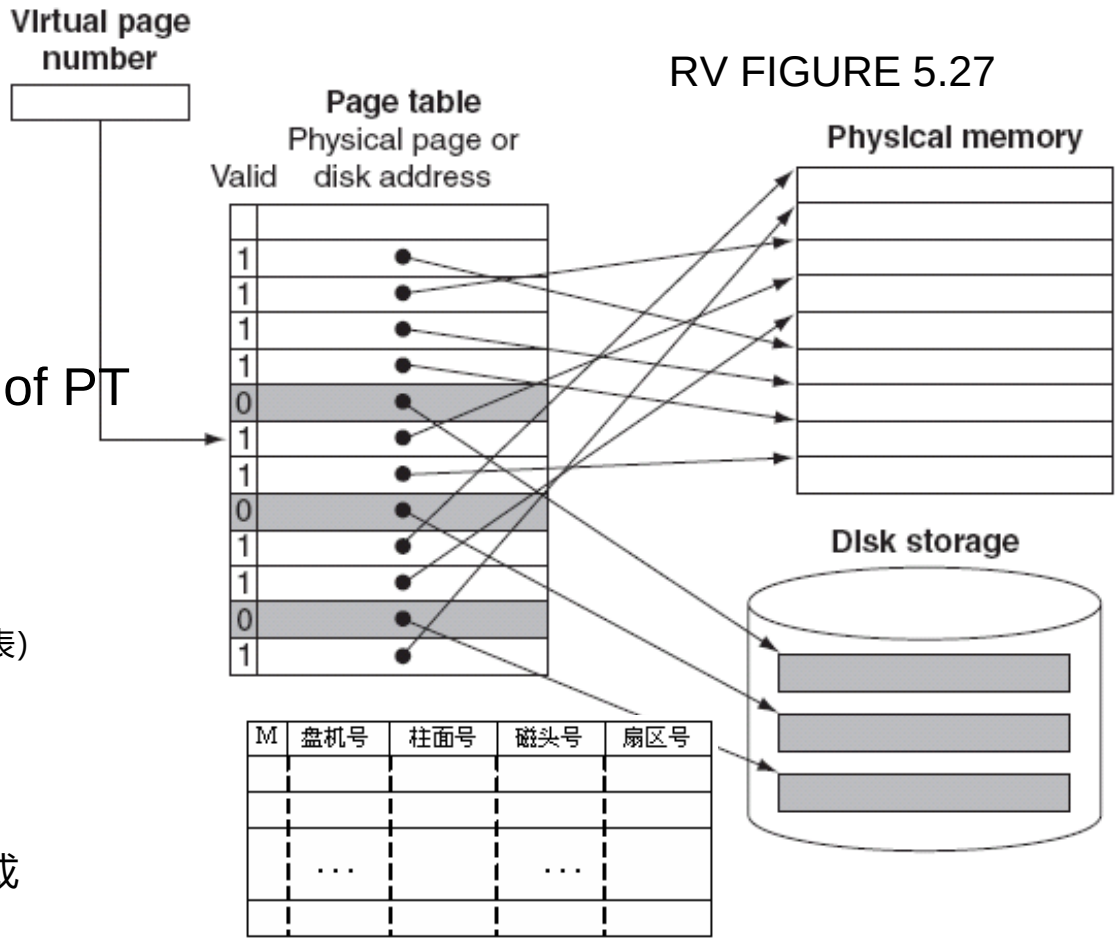
RV图5-25

页表：虚实映射，权限控制

- 由OS为进程创建和维护
  - 存储于内存中OS地址空间
    - 普通用户无法访问
  - 页表基址寄存器PTBR

- VPN（虚页号）is the index of PT
- PTE格式：page table entry
  - 控制位VCD：
    - Valid：装入，缺页异常
      - 未装入页：disk地址（外页表）
    - Ref/used/Count: $5.7.2
    - Dirty
  - 物理页号（页基址）
    - page frame number (PFN)，或
    - Disk address
  - Access Rights：非法异常$5.7.8
    - read/write/modify/exe
    - 页共享保护（限制其他进程权限）

RV FIGURE 5.27



外页表（disk map）：扇区地址

M：装入位（mount），指示是否已从第二级辅存（离线，如磁带机）中装入

- 每页size=4K。
- 虚空间大小16页（64k）
- 系统物理内存4页（16K）

- 页表：16项=虚空间大小
  - 每个entry对应虚空间的一个页
  - Frame域：物理页框号

本例，虚存的第0页：
逻辑地址0000H~0FFFH
存储在内存的1号页框中，物理地址为1000H~1FFFH

例：虚拟地址＝0xxxH

虚页号　　页内偏移

	Frame	Valid	Count	Dirty
0	0 1	1	1 0	0
1	0 0	0	0 0	0
2	0 0	0	0 0	0
3	0 0	0	0 0	0
4	1 1	1	0 1	1
5	0 0	0	0 0	0
6	0 0	0	0 0	0
7	0 0	0	0 0	0
8	0 0	0	0 0	0
9	0 0	0	0 0	0
A	0 0	0	0 0	0
B	0 0	0	0 0	0
C	0 0	0	0 0	0
D	0 0	0	0 0	0
E	0 0	0	0 0	0
F	0 0	1	0 0	0

(a)

（a）为页表
（b）为对应的物理内存

物理页框
•3
•2
•1
•0

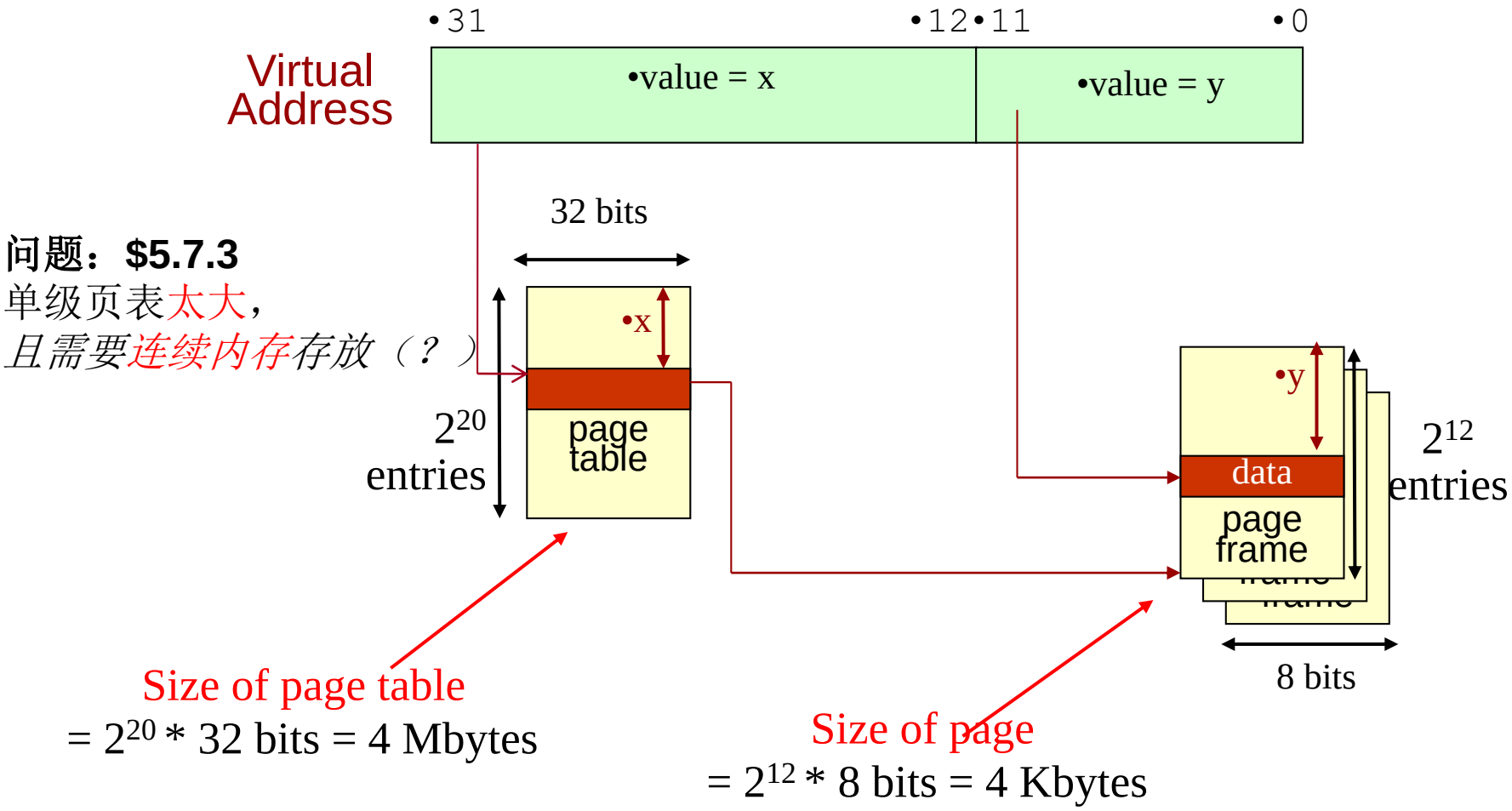
装入的虚存页号

3FFFH 3000H	Page 4
2FFFH 2000H	Unused
1FFFH 1000H	Page 0
0FFFH 0000H	Page F

(b)

•如果访问虚存page “A”？
•执行一条访存指令需访问几次存储器？

Single-Level Page Table



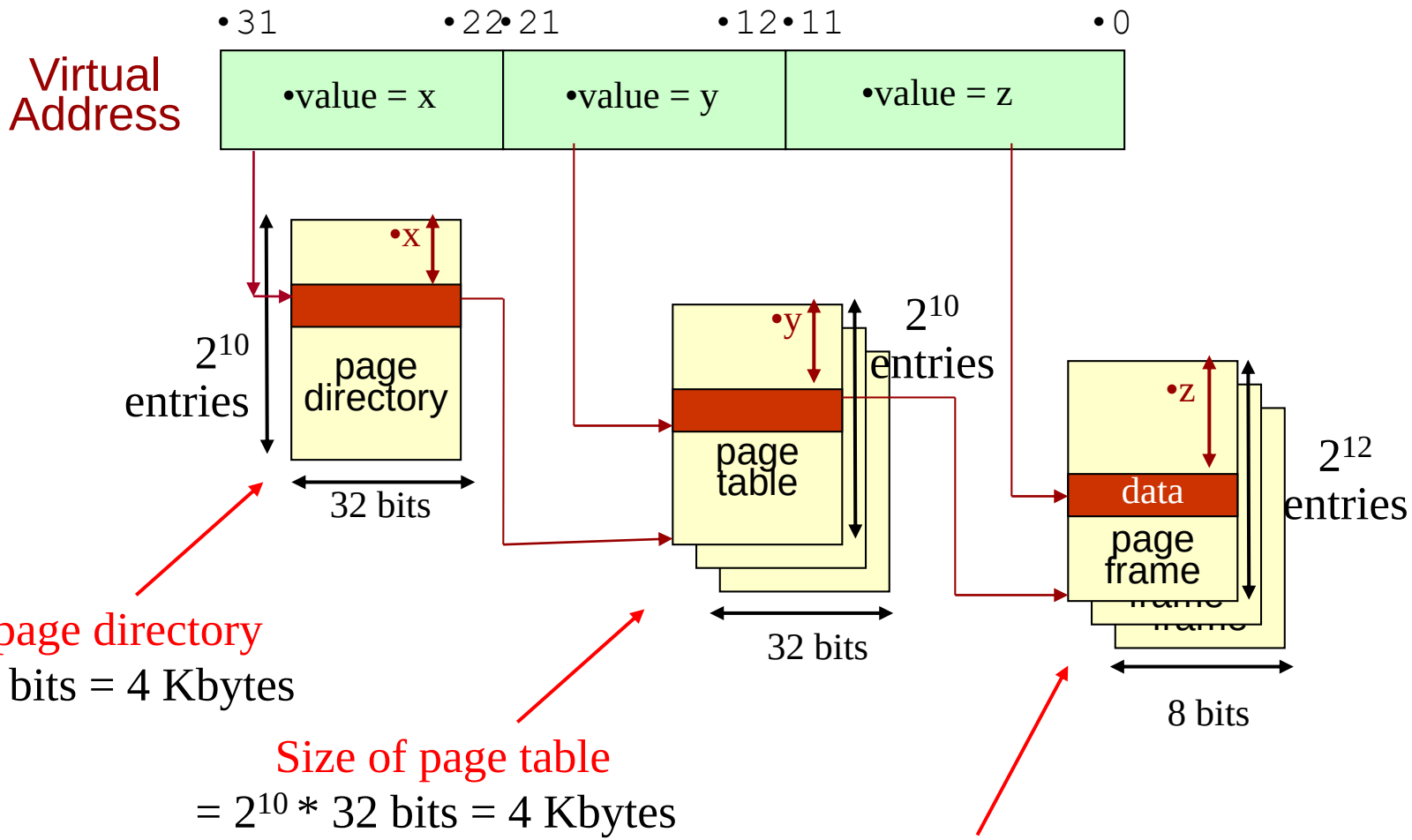
问题：$5.7.3

单级页表太大，

且需要连续内存存放（？）

例2：48-bit canonical form virtual addresses and 4 KB pages. On such a system, it would take 2^48 B / 2^12 B * 8 B = 2^39 B = 512 GB of storage just for the page table alone!

Two-level Page Table



Size of page directory
= 2^{10} * 32 bits = 4 Kbytes

Size of page table
= 2^{10} * 32 bits = 4 Kbytes

Size of page
= 2^{12} * 8 bits = 4 Kbytes

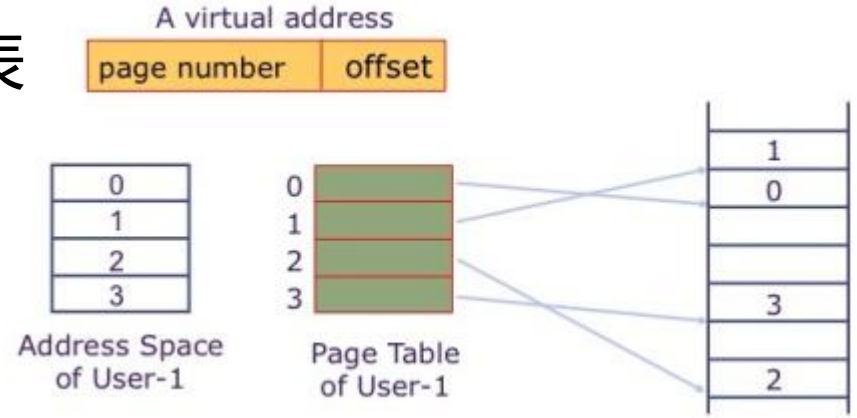
多级页表：$5.7.3

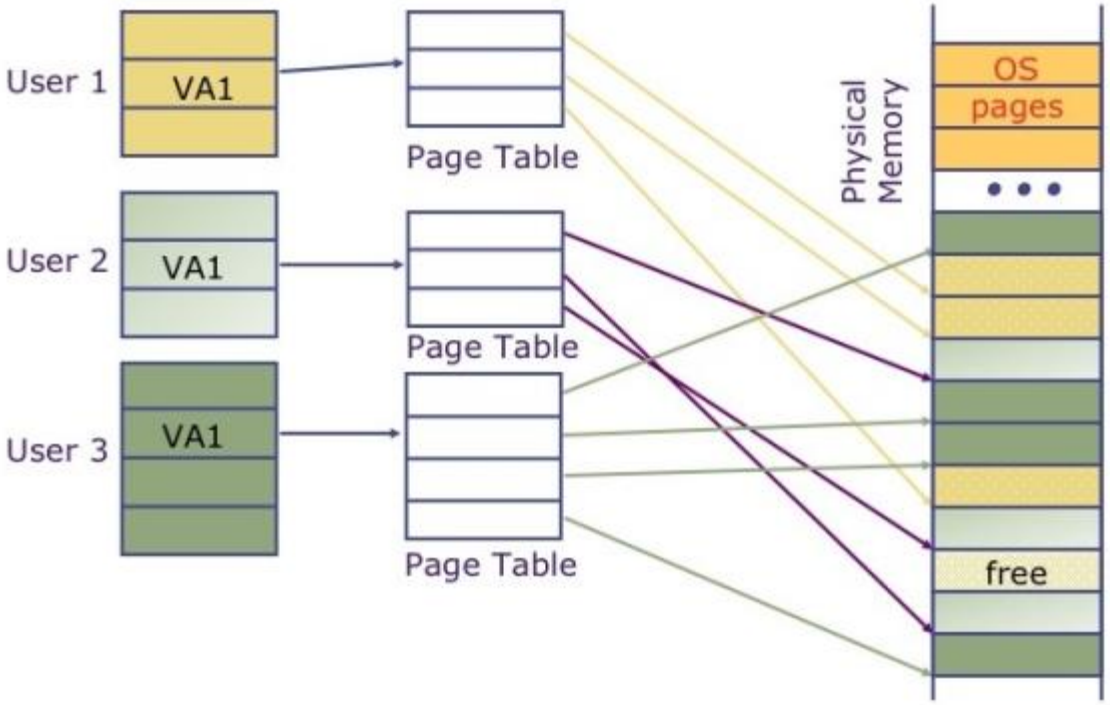
1）按需调入内存——*后续讨论忽略！*

2）可分别置于非连续的内存存储区

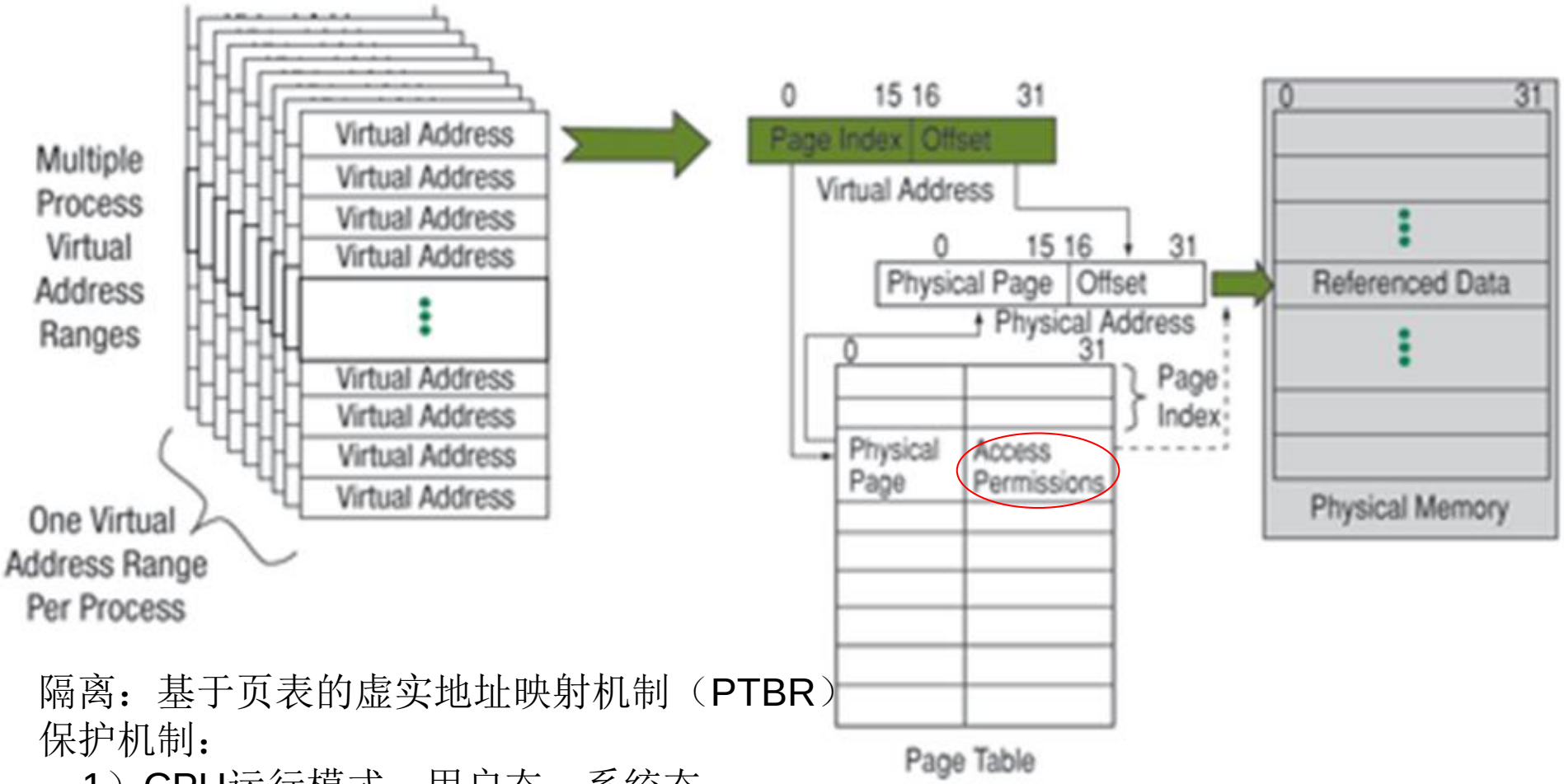
多进程管理：$5.7.8

- OS为每个进程创建和维护页表
- 进程状态
  - PC+GPRs+SP+PTBR
- 进程切换：
  - 保存上下文，恢复上下文





多进程：隔离，保护，共享



隔离：基于页表的虚实地址映射机制（PTBR）

保护机制：

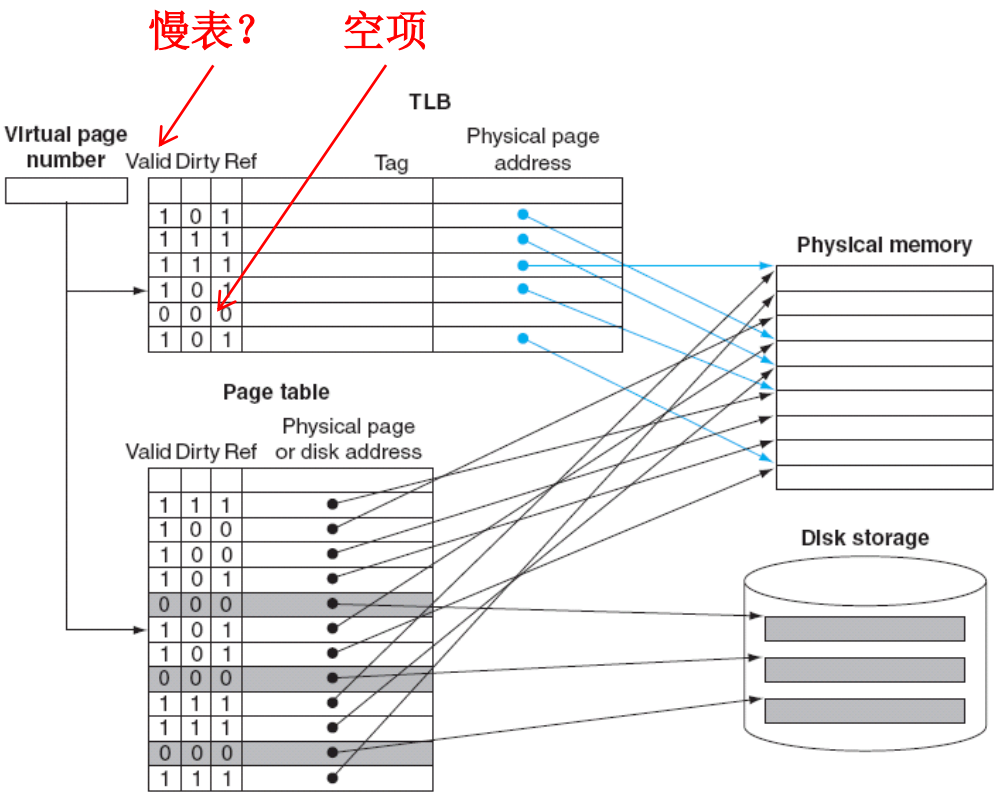
1）CPU运行模式：用户态，系统态；

2）页表位于OS地址空间，防止用户进程修改，只能由OS维护；

3）页访问权限位（Access rights）：只能OS更改。非法访问异常。

共享：OS在P1的页表中建立一个指向P2页的PTE，允许P1访问P2。ASID（进程id）

快表TLB(Translation Look-aside Buffers)，$5.7.5



•RV图5-29

慢表项何时进入TLB？

- TLB表项
  - tag+PTE(VCD，PN，Access)
    - Tag = VPN
    - 多级页表：信息来自最后一级慢表
  - 映射方式：fully associative
  - 替换：随机/FIFO（简单）
- 查找：比较Tag+V?
- TLBhit：更新ref和dirty位。
  - 标志位可能与慢表不一致
    - 写回：表项换出时写回VCD
- TLBmiss：TLB表项inv
  - CPU stall
  - 查慢表（ Table walking ）
    - PageFault：慢表项inv，Ctx Switch
  - CPU继续或指令重启（$5.7.9）
- 进程切换：强制清空TLB
  - 减少清空开销：ASID（进程ID）

CPU访存过程：虚存、TLB、Cache

- 实地址Cache：执行一条load指令，需要访存几次（BC，WC）？
- *虚地址Cache：快；别名（aliasing）问题*

物理Cache：

顺序访问TLB和Cache

可流水化（$5.7.7详解）

逻辑Cache：

TLB在Cache后？

TLB miss：MMU或OS异常

RV$5.7.6：MIPS由软件处理

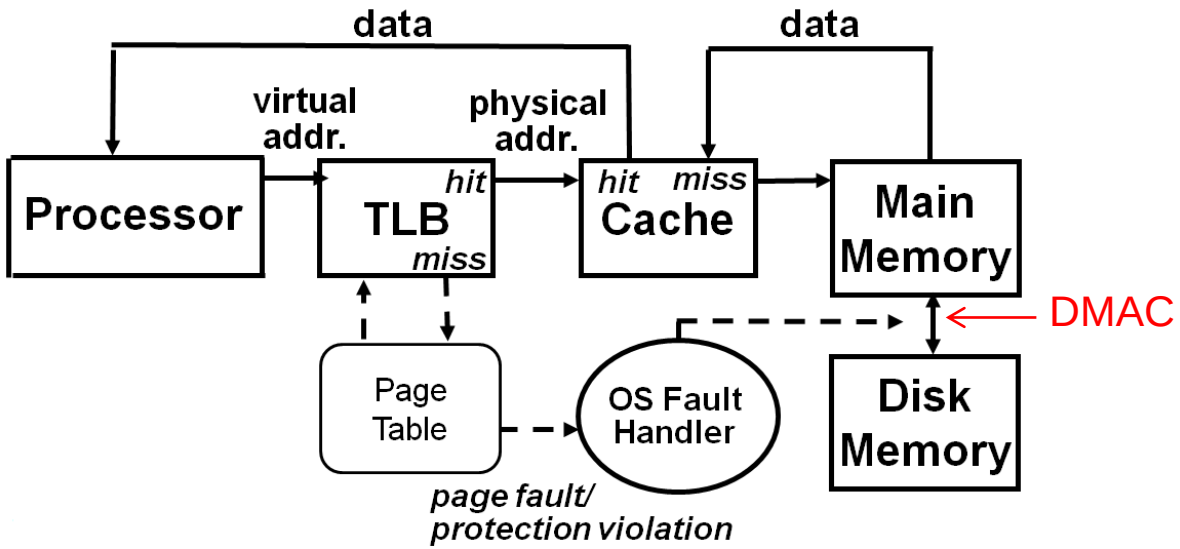
RV？

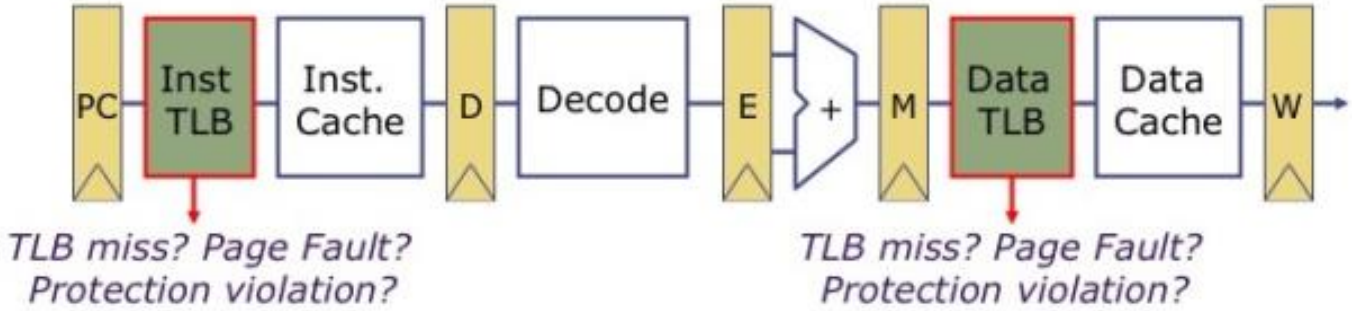
缺页：OS异常

非法：OS异常

access保护违例

PPL：时钟周期宽度？





FastMATH的TLB

TLB表项64位：
=VPN20+PN20+VCD
+V+Access+ASID

TLBhit：tag+V

注意物理地址格式
内存页大小
Cache块大小

对照图5-12的Cache（直接映射）

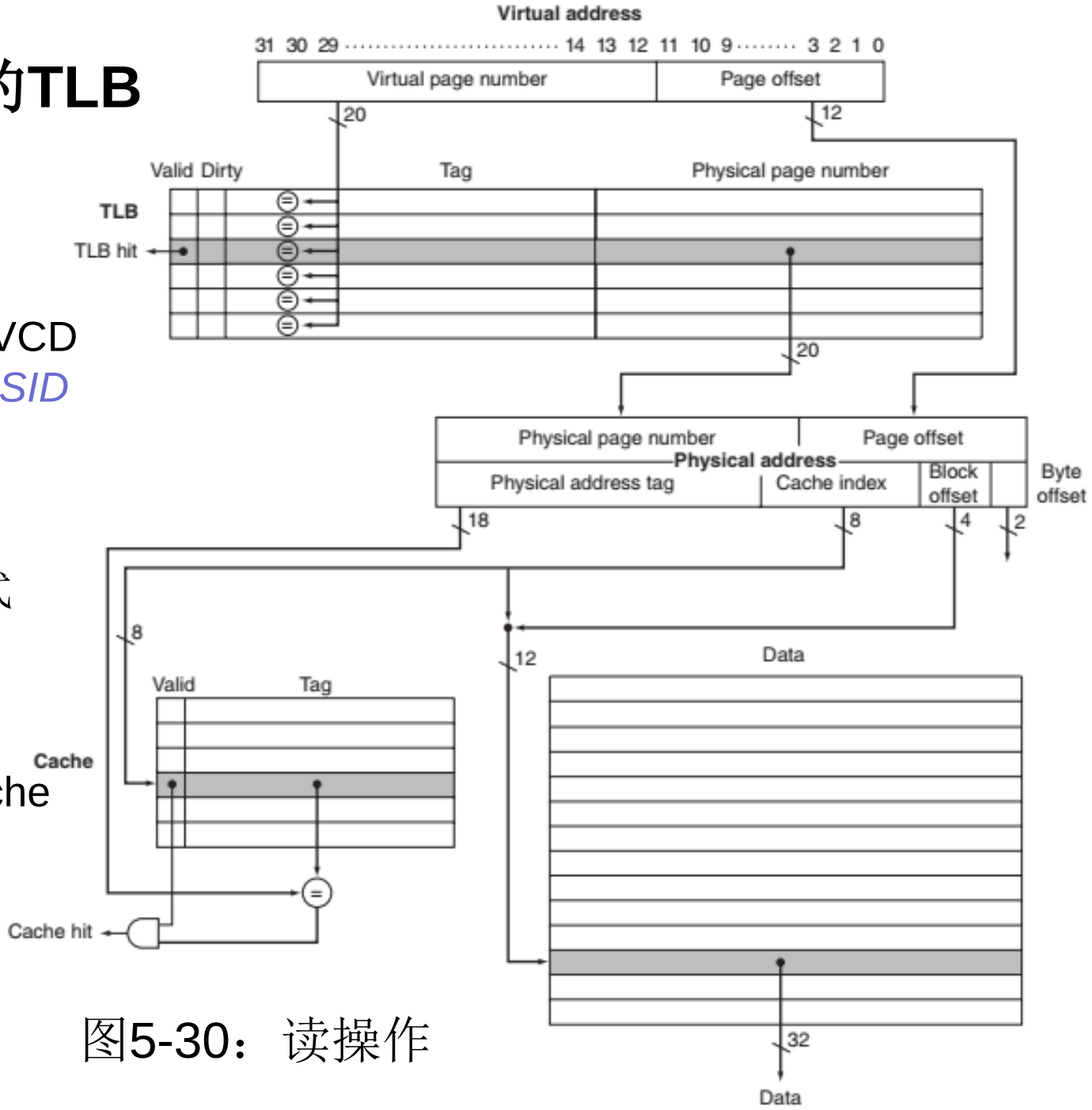
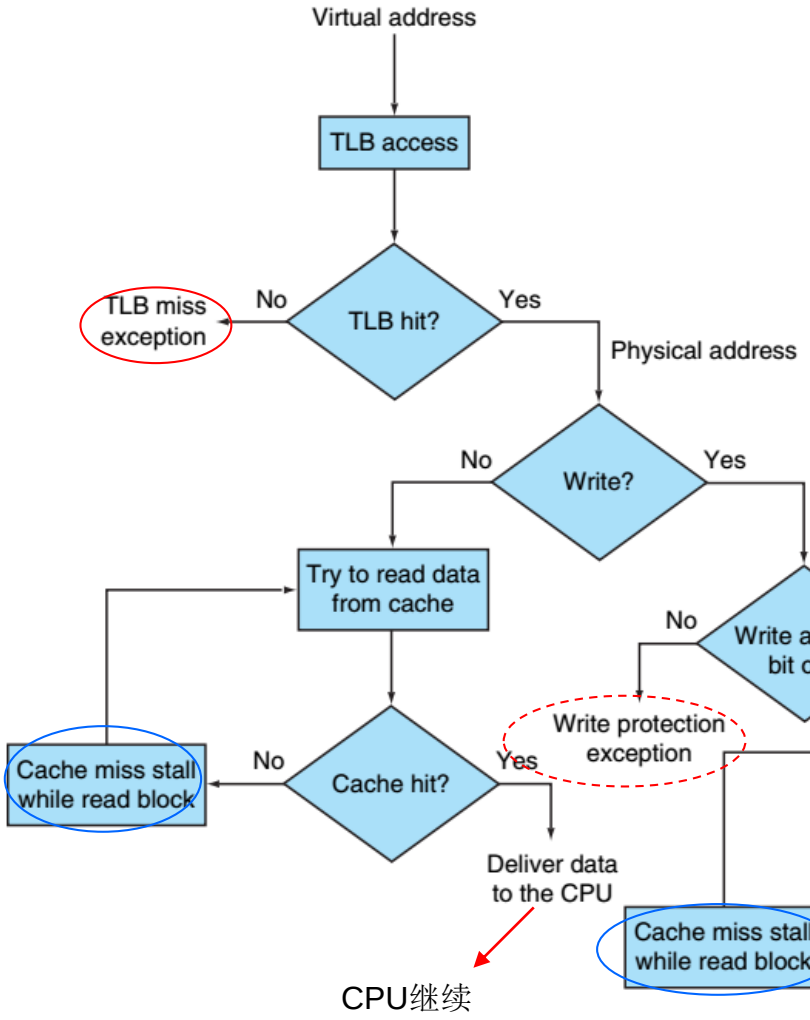
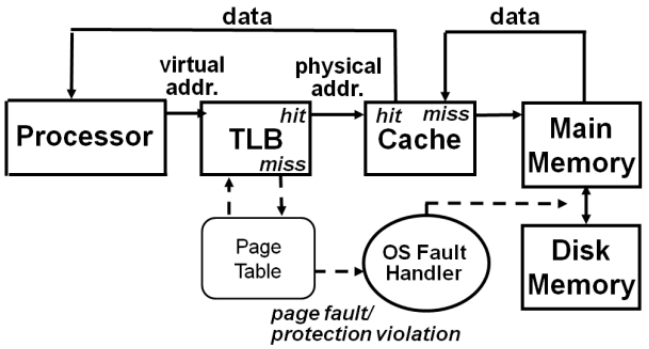


图5-30：读操作

访存流程图：TLB（MMU），Cache，OS



- TLB
  - 命中
    - 权限（access rights）错：OS非法访问异常
  - TLB miss：无VPN Tag，*或无效?*
    - 查慢表：硬件（MMU）？软件（MIPS）？
      - 慢表命中：返回PN，更新TLB（随机替换）
        - » MIPS异常处理“约13 cycles”，$5.7.6
      - 慢表Invalid或权限错：非法访问异常
- 注意
  - 此流程假设慢表命中（无缺页异常）

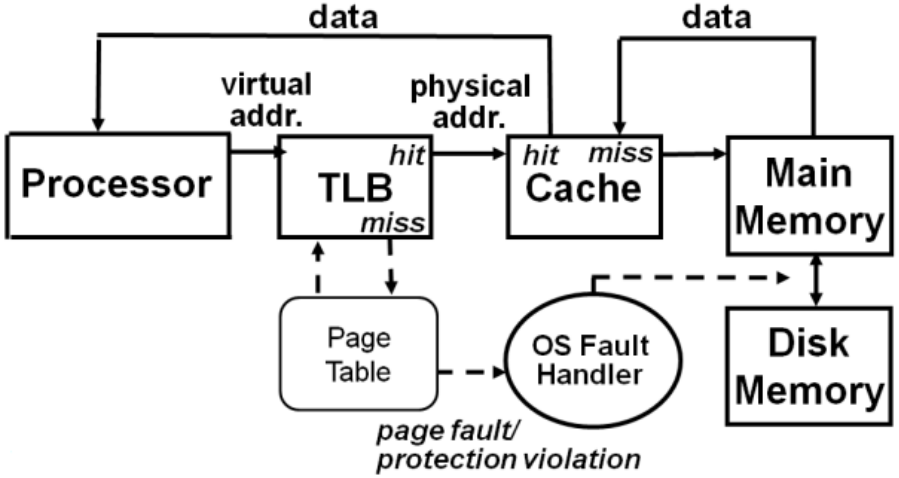


FastMATH Cache：
写透，写分配，writebuf

RV图5-31： 针对FastMATH

访存异常（RV图5-32）

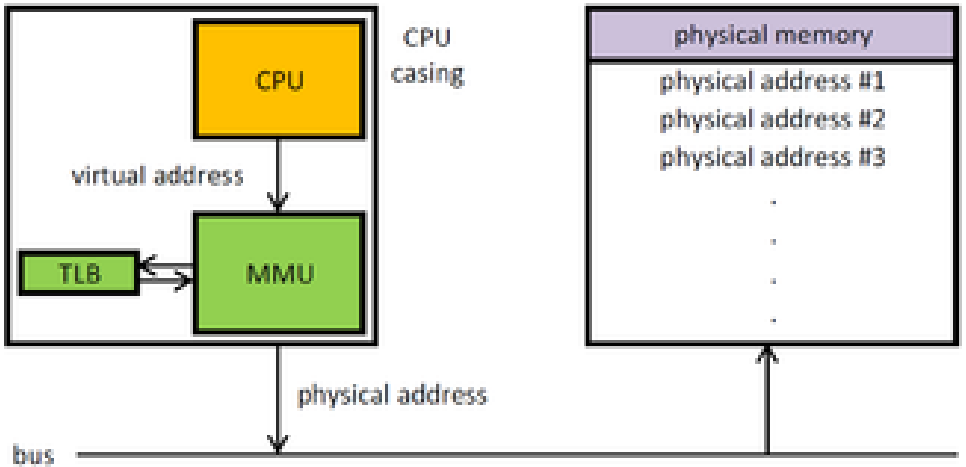
- 一次访存可能发生三种miss
  - TLB miss：TLB表项（entry）为invalid
  - Cache miss：Cache控制器处理
  - mem miss：page fault（invaild）
- 三种组合“不可能”
  - TLB命中，则不可能缺页
  - TLB miss，则不可能Cache命中
- 注意：此表为物理Cache



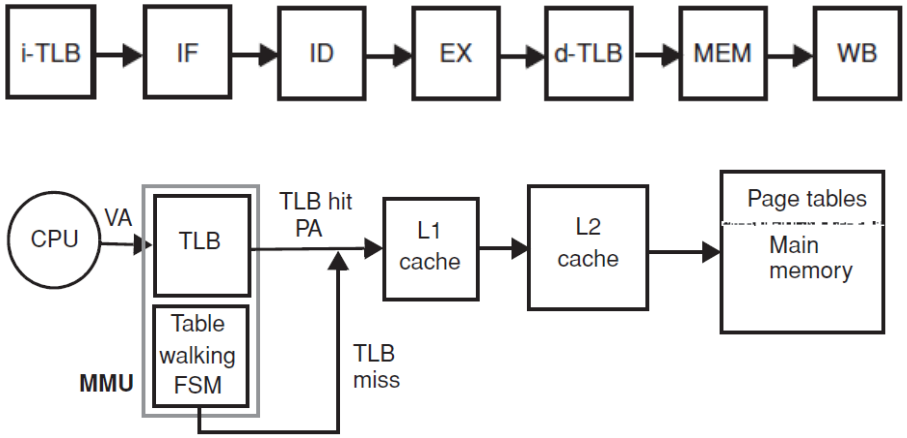
TLB	Page table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	Possible, although the page table is never really checked if TLB hits.
miss	hit	hit	TLB misses, but entry found in page table; after retry, data is found in cache.
miss	hit	miss	TLB misses, but entry found in page table; after retry, data misses in cache.
miss	miss	miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
hit	miss	miss	Impossible: cannot have a translation in TLB if page is not present in memory.
hit	miss	hit	Impossible: cannot have a translation in TLB if page is not present in memory.
miss	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory.

MMU（Memory Management Unit）

- 功能：MAPPER（Table walking）
  - 虚实地址转换：不同进程的虚空间隔离
    - MMU被禁止时，虚地址直接输出到物理地址总线
  - 访问控制：权限
    - 共享内存：检查指令对当前页的访问权限，别名
  - MMU fault异常请求：缺页，非法访问，转换错，…
- 组成：*TLB*，页表基址寄存器PTBR，AccessPermisionControl，*FSM*



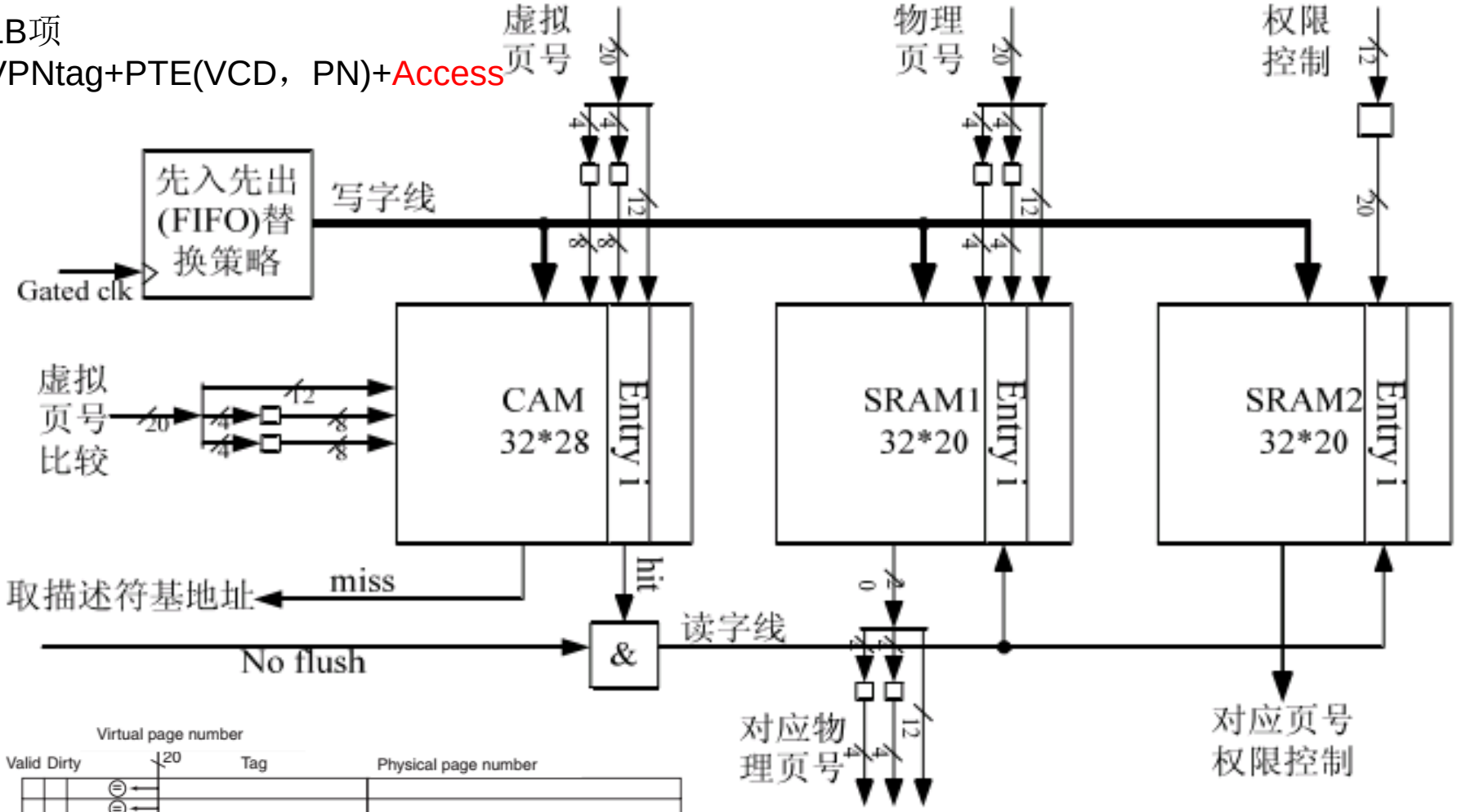
CPU: Central Processing Unit
MMU: Memory Management Unit
TLB: Translation lookaside buffer

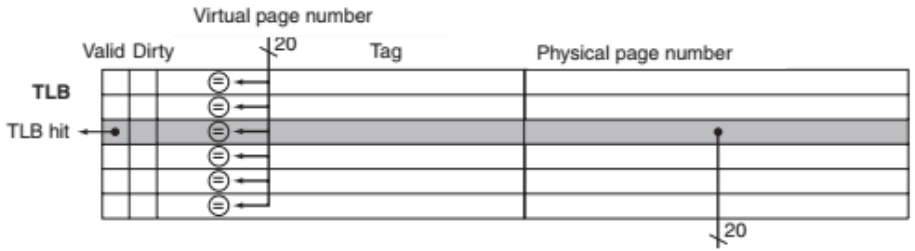


Dubois，《并行计算机组成与设计》，2017

MMU实现：数据通路，全相联TLB

TLB项
=VPNtag+PTE(VCD，PN)+Access

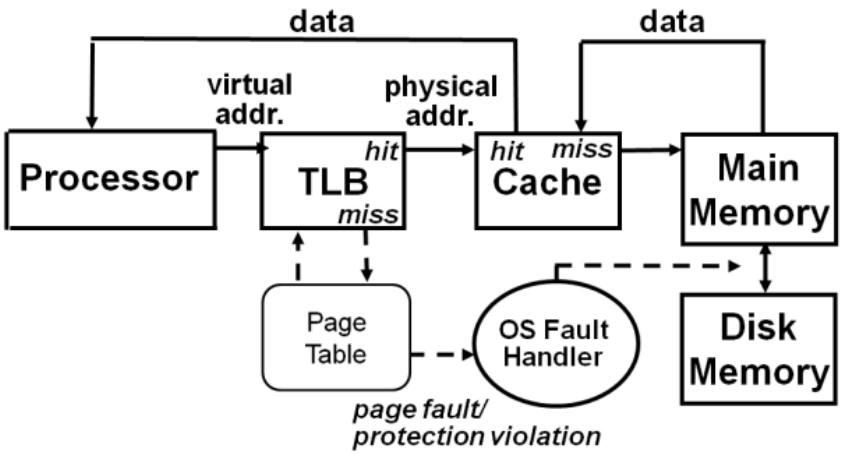




- Hit time: 0.5~1 clock cycle
- Miss penalty: 10~100 clock cycles
- Miss rate: 0.01%–1%

MMU实现：Table walking FSM?

- TLB表项维护：查慢表，VCD更新，替换
  - 参考a Direct mapped Cache Controller?
- 异常：缺页，非法
  - 指令重启
- 接口：CPU、Cache



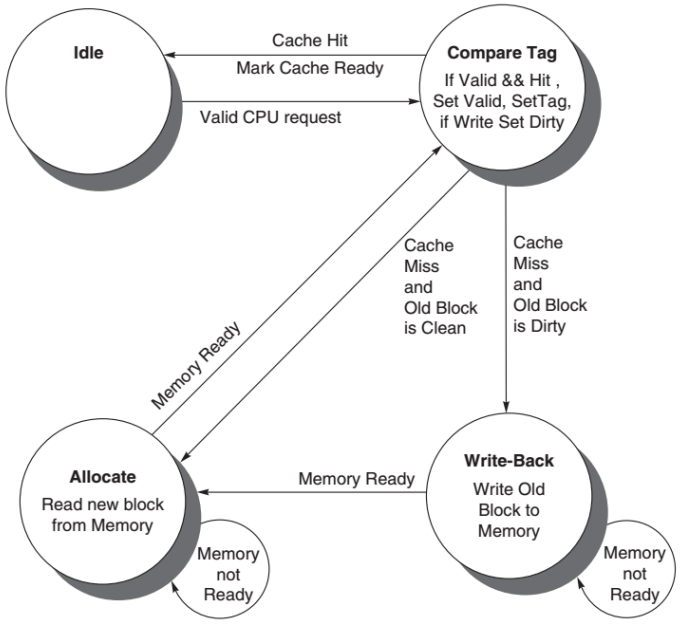
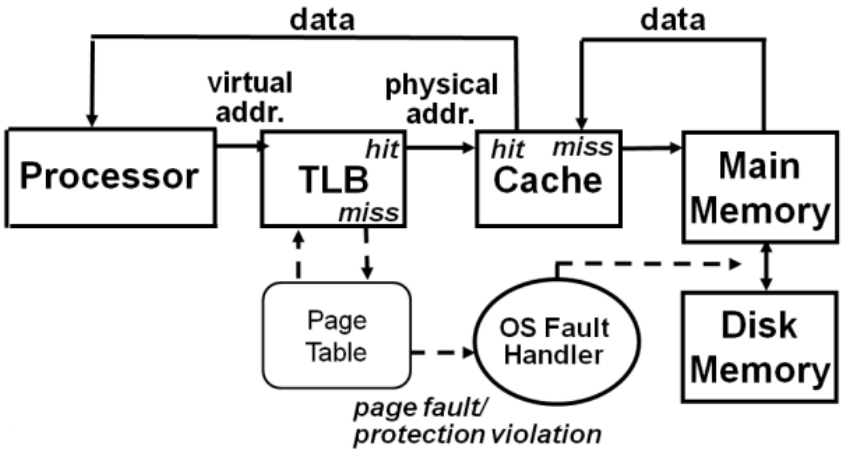


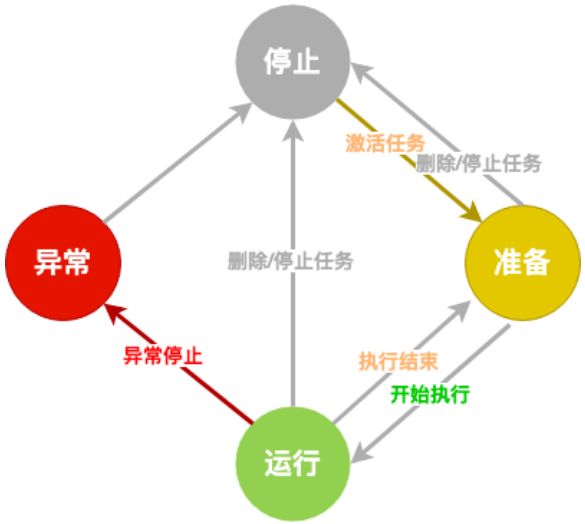
图5-38

OS的分页处理（paging）：4个时间点

Four times when OS involved with paging

1. Process creation
   - determine program size
   - create page table
   - set PTBR
   - 进程状态：PC+GPRs+PTBR（页表）
2. Process execution：Context switch
   - MMU reset PTBR for new process
   - TLB flushed：TLB的V位?
     - 减少频繁刷新：Process ID（ASID）
   - Ctx切换的条件：右下图？
   - 多级页表：访问辅存？
3. Process execution：Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
4. Process termination time
   - release page table, pages

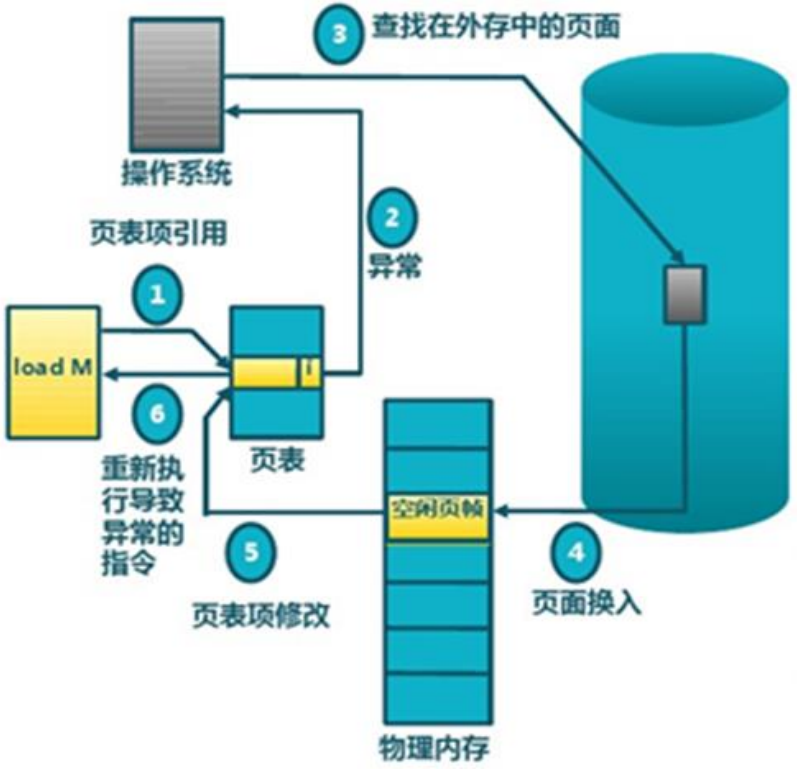




虚存管理的性能：有效访存时间

- EAT：effective memory access time
  - 访存时间*（1-P）+ 缺页处理时间 * P
    - 缺页处理时间=读盘时间+写回时间*q
    - 缺页率P，Dirty率q
  - 访存时间10ns，磁盘访问时间5ms

L1 cache reference	1 ns
Branch mispredict	3 ns
L2 cache reference	4 ns
Mutex lock/unlock	17 ns
Main memory reference	100 ns
Send 2KB over commodity network	250 ns
Compress 1KB with zip	2 us
Read 1MB sequentially from main memory	9 us
SSD random read	16 us
Read 1MB sequentially from SSD	156 us
Round trip in datacenter	500 us
Read 1MB sequentially from disk	2 ms
Disk random read	4 ms
Packet roundtrip from CA to Netherlands	150 ms



可用内存大小

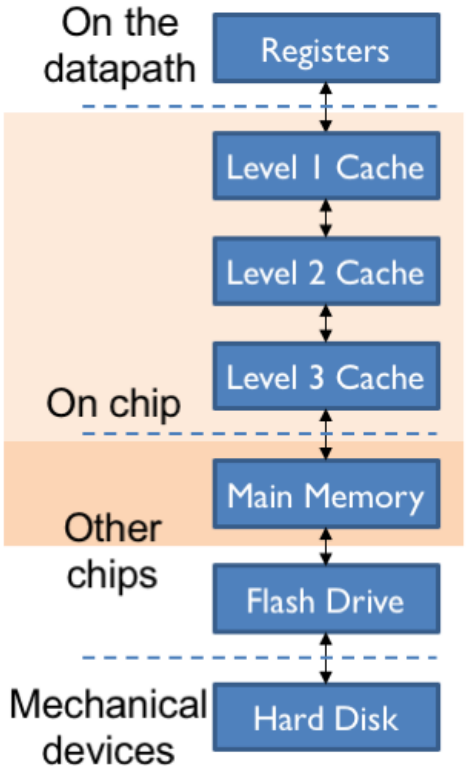
用free -m查看的结果：

[root@localhost ~]# free -m

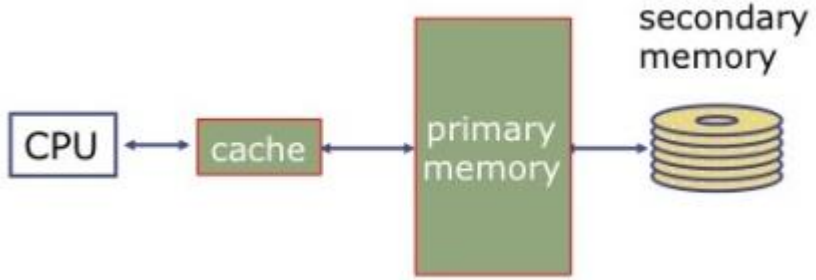
	total	used	free	shared	buffers	cached
Mem:	7918	7865	52	0	7228	143
-/+ buffers/cache:		493	7424			
Swap:	4996	0	4996			

- 用 used 减去 buffer 和 cache，才是运行中的程序所占用的空间

Everything is a cache of others

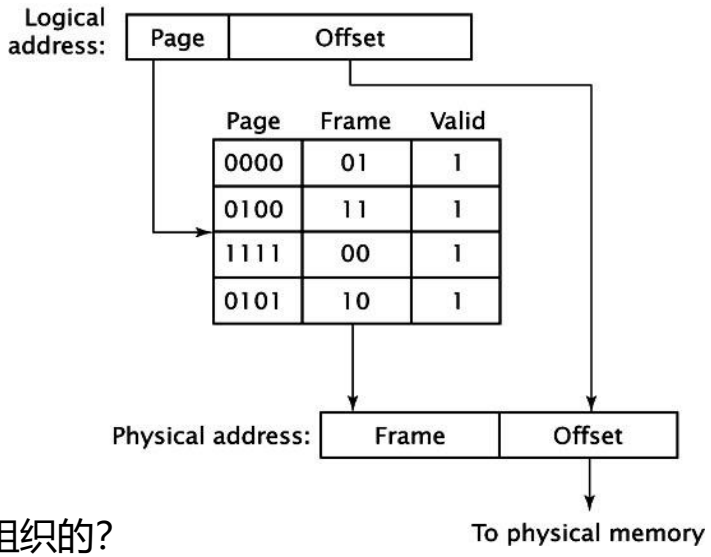


Access time	Capacity	Managed By
1 cycle	1 KB	Software/Compiler
2-4 cycles	32 KB	Hardware
10 cycles	256 KB	Hardware
40 cycles	10 MB	Hardware
200 cycles	10 GB	Software/OS
10-100us	100 GB	Software/OS
10ms	1 TB	Software/OS



小结

- 实方式下，程序空间=物理空间
- 虚方式下，每个进程有自己的虚空间
  - 虚存体系的主要作用?
  - 每个进程一个（套）页表，由OS创建进程时创建
  - 在辅存中保存进程的虚空间，主存为虚空间的部分镜像
  - 进程状态：PC+GPRs+PTBR（页表）
- 多进程的隔离与保护机制：access right
- 多级页表的功能
- 思考
  - 程序的虚地址空间与磁盘空间如何映射？swap区是如何组织的?
  - Cache与MMU实现机制比较?
  - 程序执行前，OS要做什么?
  - 系统启动时是实地址方式，何时转为虚方式？切换时发生了什么?
  - 虚存管理哪些由硬件实现，哪些由OS实现?
    - MMU中包含哪些模块?
  - TLB miss和缺页异常如何处理?
  - 执行一条load/store指令需要访存几次？响应时间?
  - 如何实现^c和^v?
- 作业
  - RV：5.16.1，5.17.2




Thank You