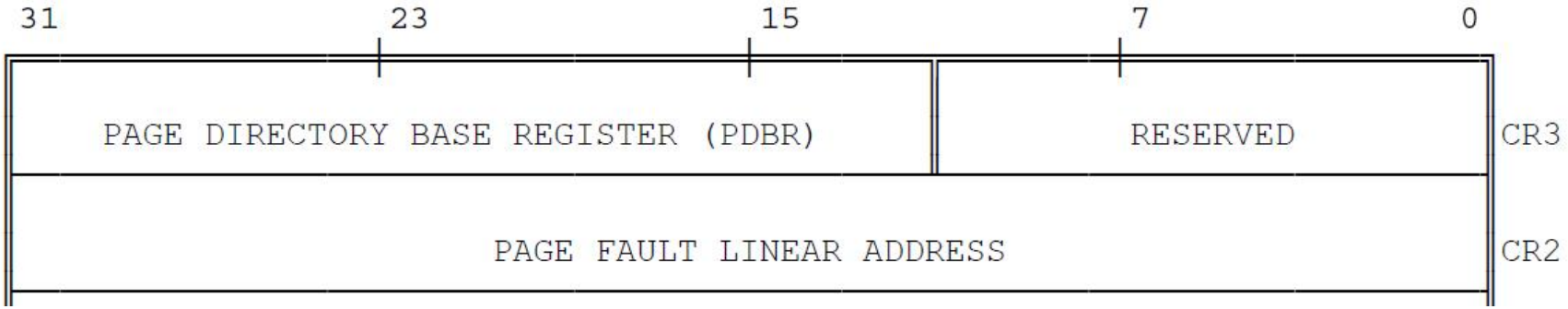
操作系统

第6章 虚拟存储器

朱小军, 教授
https://xzhu.info
南京航空航天大学
计算机科学与技术学院
2025年春

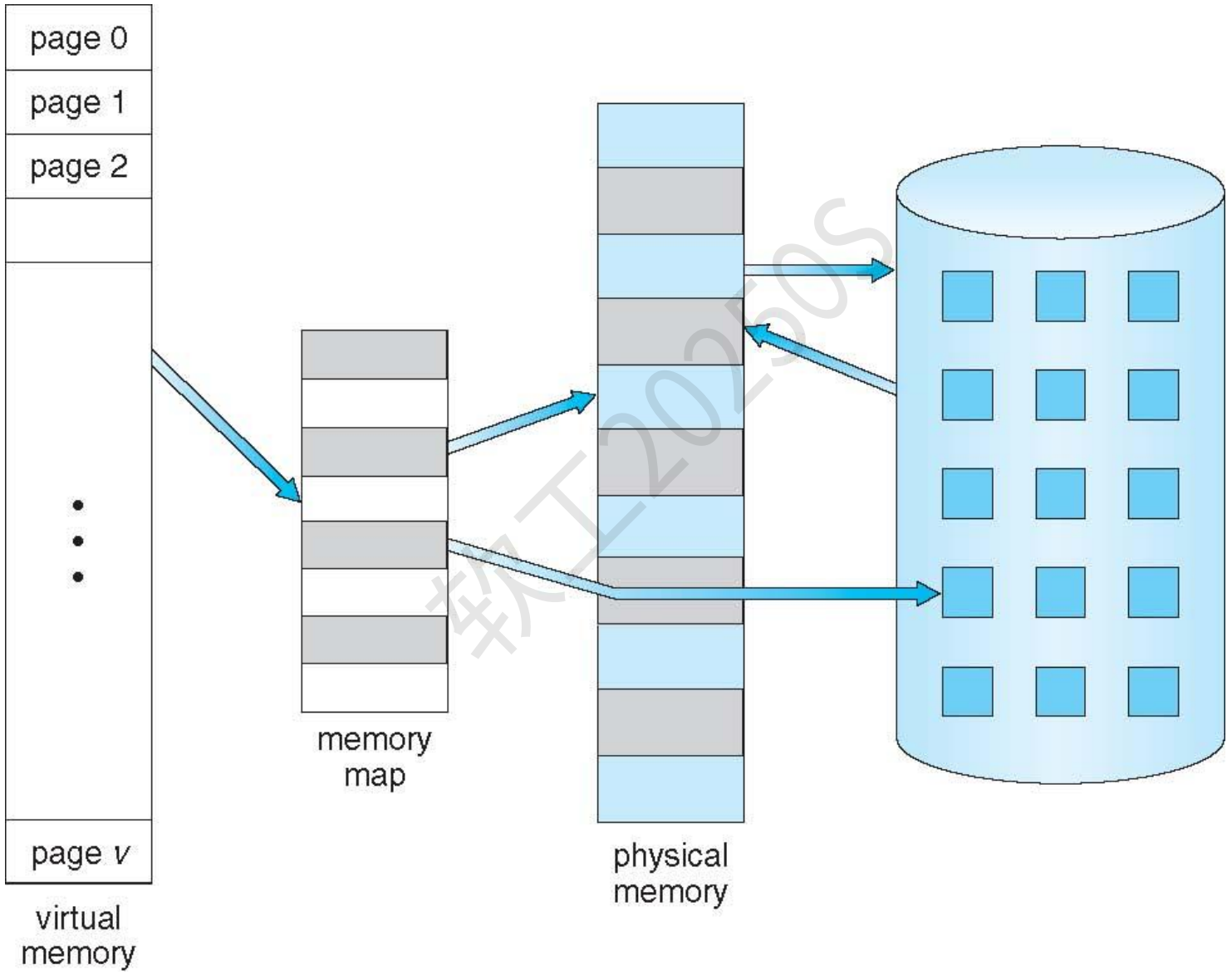
回顾：内存管理

- 给定物理内存和磁盘，满足多个进程的地址空间需求
- 物理内存管理方法
  - 固定分区 ->分区大小不好确定
  - 可变分区 ->碎片问题、紧凑
  - 分页存储管理->页表大、访存速度慢
    - 页表一个页框装不下，需要多个页框，若页框不连续，则有问题，为何？



- 分页存储管理下的若干技术
  - 代码共享
  - 共享内存
  - 修改页表后需刷新TLB，如何做？重置cr3寄存器
    - mov %eax, %cr3 <<i386>>
- 不同内存管理方法对OS的实现有何影响?
  - PCB、创建进程时、进程消亡时
- 物理内存不足如何应对?
  - 借助磁盘：对换、虚存

虚存的基本思想


page 0
page 1
page 2
page v
virtual memory
memory map
physical memory

请求分页虚拟存储器管理

- 请求分页
  - 英文可能更好理解：demand paging
  - 直到需要访问某个逻辑地址时，才分配页框
- 基本原理
  - 程序开始执行时，所有页面尚在磁盘上
  - 运行过程中根据需要为页面分配页框，调入内存
  - 如果内存已满，装入新页面时需要淘汰旧页面，便是“页面置换”问题
- 为什么是“虚”拟存储管理?

页表需要做哪些修改?

原来的页表

页号	页框号	其他信息
0	3	读
1	2	读
2	5	读写
3	16	读写
4	8	读
…	…	

- 需要指示哪些页面在内存中，哪些页面在磁盘中
- 如果在磁盘中，需要指明磁盘中的位置

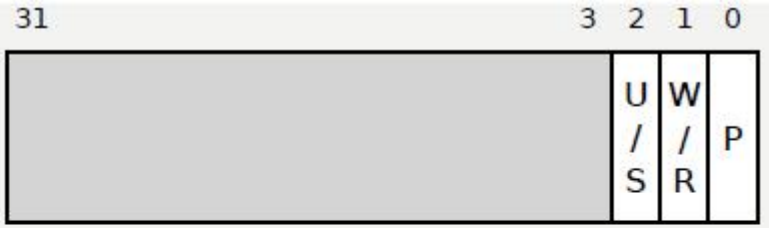
请求分页中的页表项

页面	页框	存在位	外存地址	修改位	其他信息
0		0	A		读
1	2	1			读
2	5	1			读写
3		0	B		读写
4	8	1			读
…	…	…			

- 存在位：是否在内存中
- 外存地址：不在内存中时，在外存中的位置（Linux：交换区或文件系统）
- 修改位：自调入内存中后是否做过修改（有什么用？）

地址转换与缺页中断

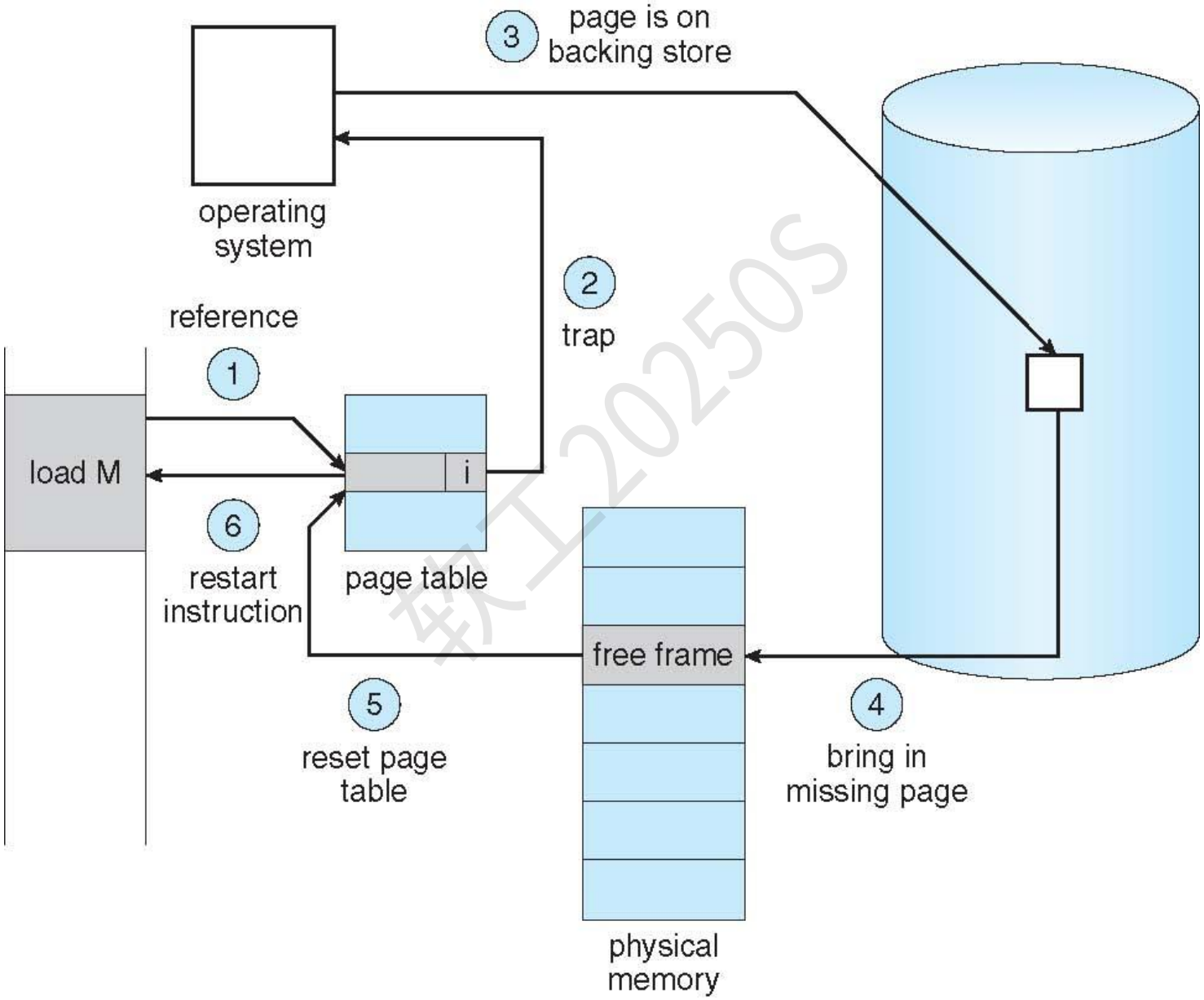
- 执行分页存储器管理的地址转换过程
- 缺页中断（硬件发出，MMU）
  - MMU若发现页面不在内存(how?)，发出缺页中断
  - 操作系统处理中断，从外存中(where?)调入页面
  - 重新执行原来的指令（而不是下一条指令）
- x86下，缺页中断的error code
  - xv6的trapframe的err
  - err为5指什么?



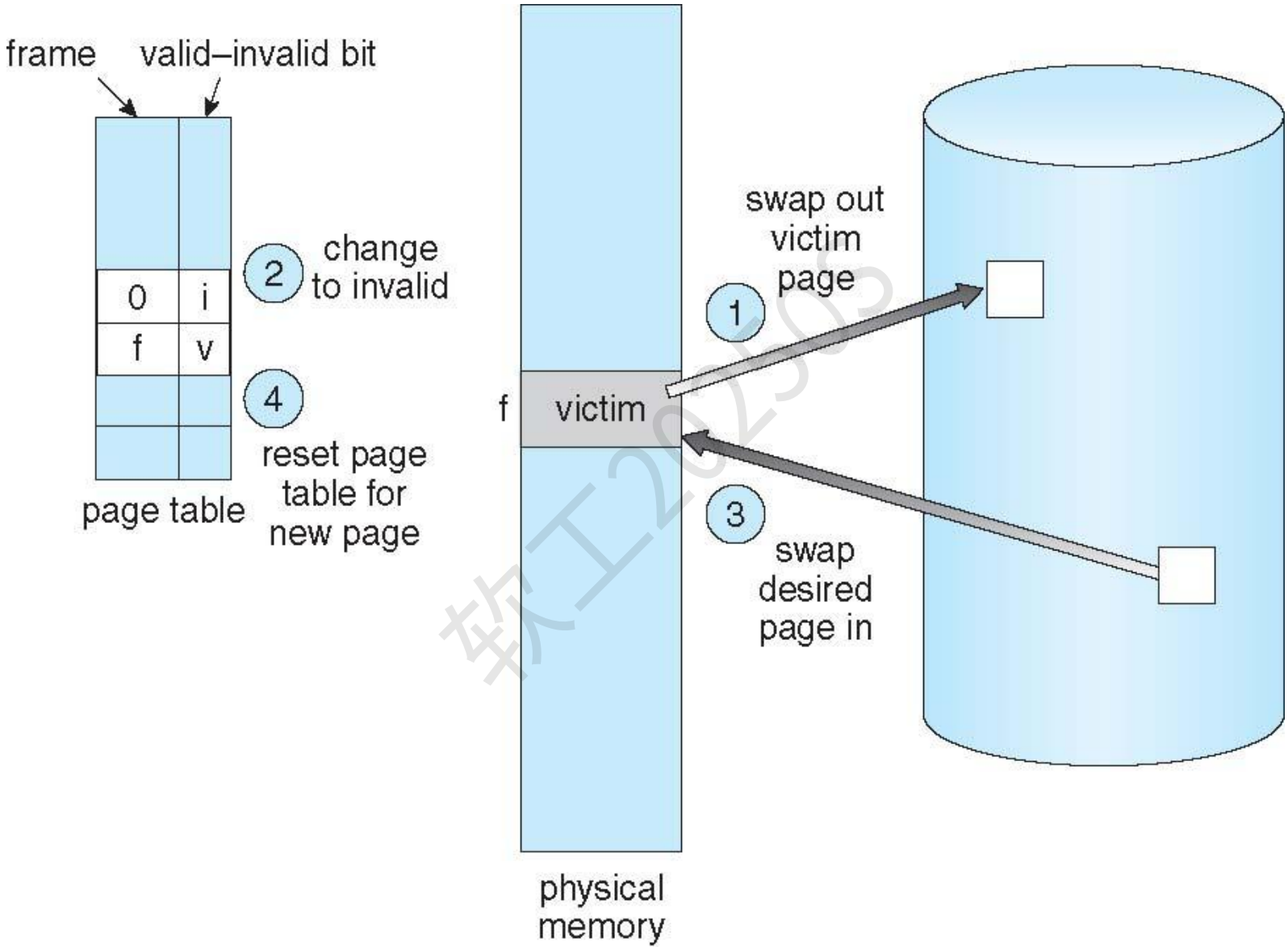
P	0=Non-present page 1=Protection-violation
W/R	Cause (0=Read, 1=Write)
U/S	Mode (0=Supervisor, 1=User)

- 缺页中断由哪个地址引起?

处理缺页中断



缺页中断时，若需要分配页框，但系统没有空闲的，应如何处理?


frame
valid–invalid bit
0
i
f
v
page table
2
change
to invalid
4
reset page
table for
new page
f
victim
1
swap out
victim
page
3
swap
desired
page in
physical
memory

页面置换算法

（最佳置换、FIFO、LRU、Clock、改进的Clock、工作集）

为什么要置换页面?

- 因为页框不够用!
- 问题建模
  - 给定一定数量的页框，设计一个页面置换策略使得缺页率最低。
  - 缺页率：访问页面失败的比例
  - 注意：这个模型中的一部分信息是“未知”的，即未来访问页面的次序（why？）
- 避免产生“抖动”
  - thrashing
  - 刚被换出的页面又要被访问

例子：假设只有3个页框，则下列对页面的访问必然造成页面置换，应该置换哪些页面？

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

【注：实际中不可能是这样，why？】

最佳置换算法

- 假设页面的访问次序已知
- 选择未来最长时间内不再被访问的页面换出
- 对前面的例子
  - 三个页框下，页面引用次序为
  - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
  - 页面置换次序为?
- 这是一个理论上的算法
  - 实际中无法预知未来的情况（似曾相识？）
  - 但可以“事后”运行，用于性能对比

先进先出算法（FIFO）

- 淘汰最先分配页框的页面
  - 类比：超市货架不够用时。。。
  - 简单，但会淘汰频繁重复被访问的页面，如全局变量、部分代码等的页面
- 分配的页框数增加,有时缺页中断反而增加
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
  - 页框数为3时缺页中断几次?
  - 页框数为4时?
- 如何实现?

最近最久未使用（LRU）

- 将最久没使用的页面替换出去

例子：假设只有3个页框，且有下列对页面的访问，如何置换?
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

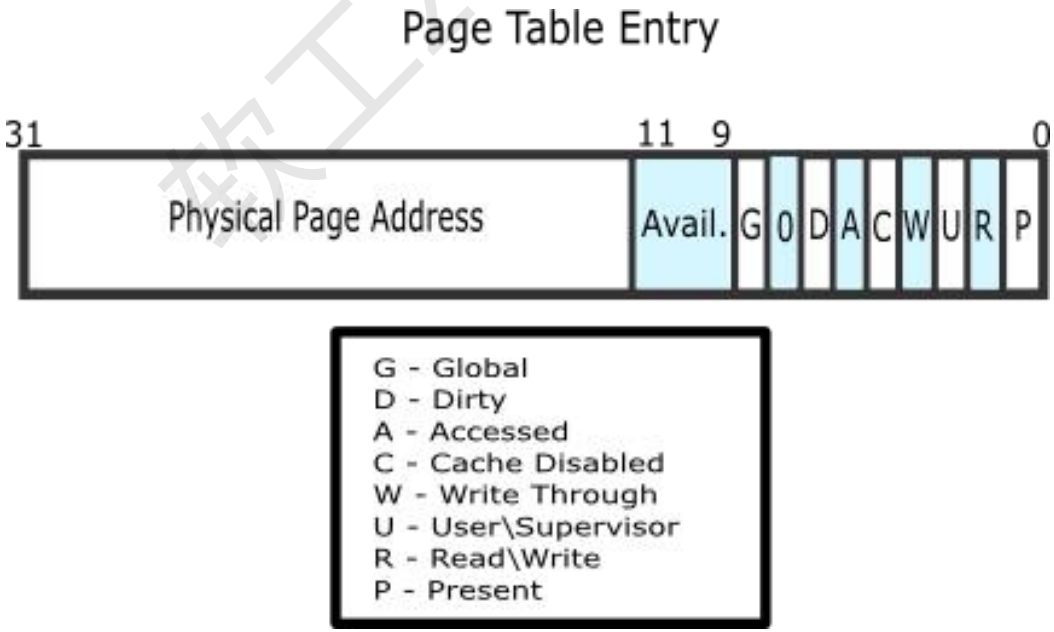
- 实现代价较高，比如：
  - 采用链表实现，频繁删除、插入，时间代价高
  - 采用计数器：每执行一条指令计数器加1；每次访存时将计数器的值存入页表项；替换时将页表项中计数最小的页面替换出去

LRU 的软件模拟实现：aging

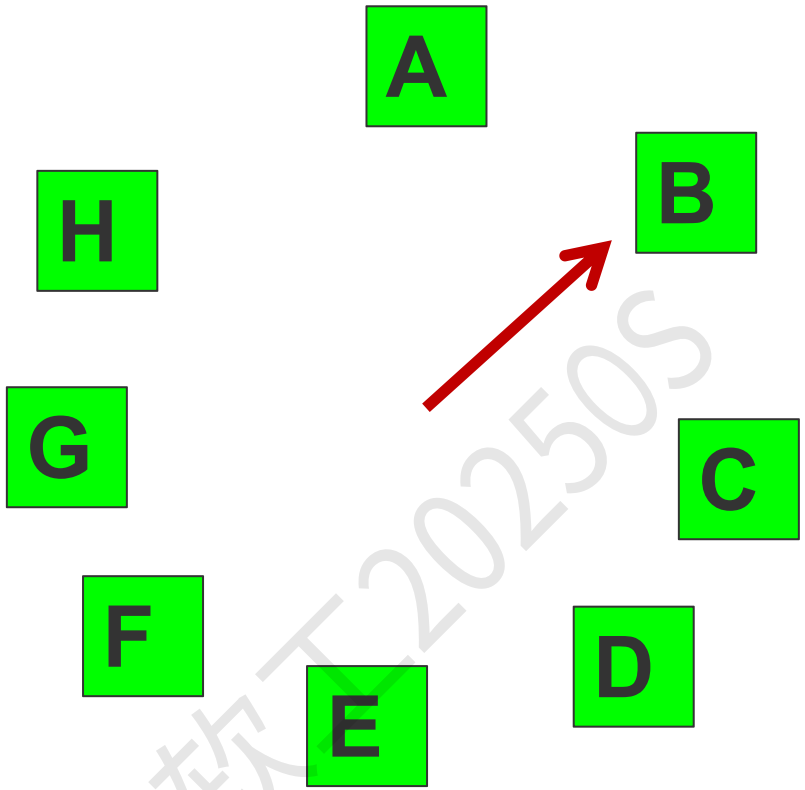
- 为每个页框设置一个计数器
  - $R=R_{n-1}R_{n-2}R_{n-3}\cdots R_2R_1R_0$
  - 每隔一段时间，时钟中断到达时，OS做以下处理
    - 将R右移一位，且，若页框被访问过（怎么知道？），最高位置1
  - 最小数值的页面即为最久未被访问的页面

实页 \ R	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
1	0	1	0	1	0	0	1	0
2	1	0	1	0	1	1	0	0
3	0	0	0	0	0	1	0	0
4	0	1	1	0	1	0	1	1
5	1	1	0	1	0	1	1	0
6	0	0	1	0	1	0	1	1
7	0	0	0	0	0	1	1	1
8	0	1	1	0	1	1	0	1

- 与LRU的区别
  - 计数器范围有限，如果两个页框的计数器均为0，则无法区分
  - 无法区分时钟周期之间的访问次序
- 如何实现?



Clock 算法



- 每个页面设置一个状态位 A，若被访问则置1
- 当发生缺页中断时，检查头指针指向的页面
  - 若A=0，则换出；否则，设置A=0，检查下一页

都被访问过的页面给了一样的待遇

- 被修改过的表示不服！

  “他是被读过，又不是被写过，淘汰它不需要写回磁盘！”

改进型Clock 算法

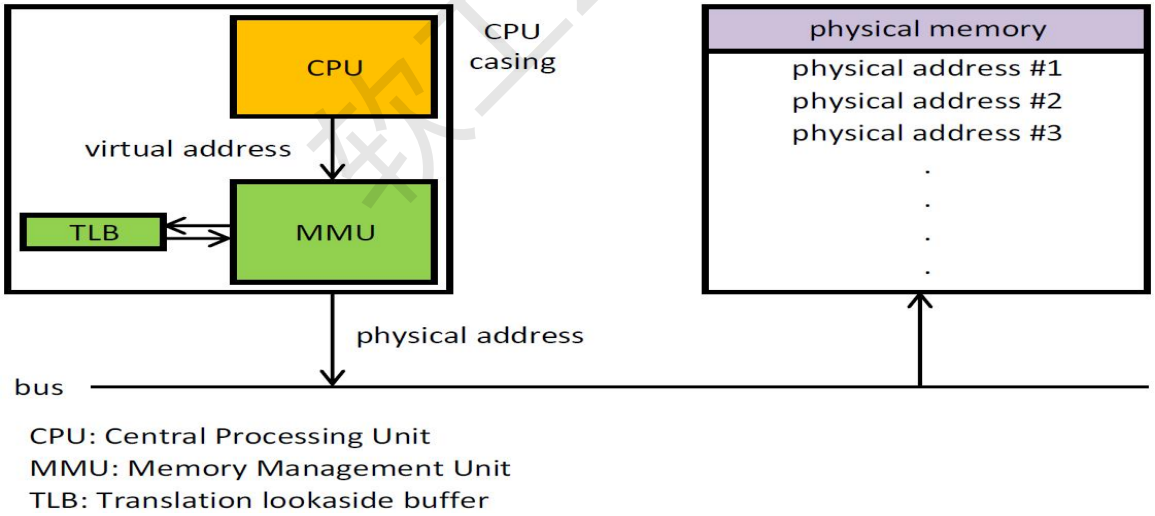
- 每个页面设置两个状态位
  - A（accessed，被访问过）、M（modified，被修改过，dirty）
- 当发生缺页中断时，淘汰次序
  - 00 01 10 11（疑问，怎么会有01？因为算法清A）
- 算法流程
  - 第一轮：寻找一个A=0，M=0的页面，淘汰，结束；
  - 第二轮：寻找A=0且M=1的页面，淘汰，结束，否则将遇到的页面的A清零；
  - 第三轮：进入第一轮，最多再进入第二轮。

回顾：虚拟内存管理

- 虚拟内存的基本概念
  - 初始不分配页框
  - 执行时报错（缺页中断），再把相应的页分配页框，其他页依然不分配
  - 万一运行中没有空闲页框怎么办？
- 页置换算法
  - 最佳置换
  - LRU
  - Clock算法
  - 改进型Clock算法

缺页中断的硬件基础知识

- 知识点
  - MMU、TLB、CPU集成在一块芯片上
  - MMU发出的地址放到总线上
  - MMU做地址转换时查询TLB
  - CPU发出一个地址，如果TLB命中，MMU会往地址线上放一次地址；如果TLB未命中，得放3次（+页目录、页表）



https://en.wikipedia.org/wiki/Memory_management_unit

课前提问

这些地址是逻辑地址还是物理地址？

```
$ threadtest

------------------- Test Return Value -------------------
unexpected trap 14 from cpu 0 eip 80104551 (cr2=0xcff8)
lapicid 0: panic: trap
 80106[illegible] 80105e2c 80105d53 80104f39 [illegible]0bd 80105e2c 0 0 0 0
```

A :某时刻eip寄存器的值

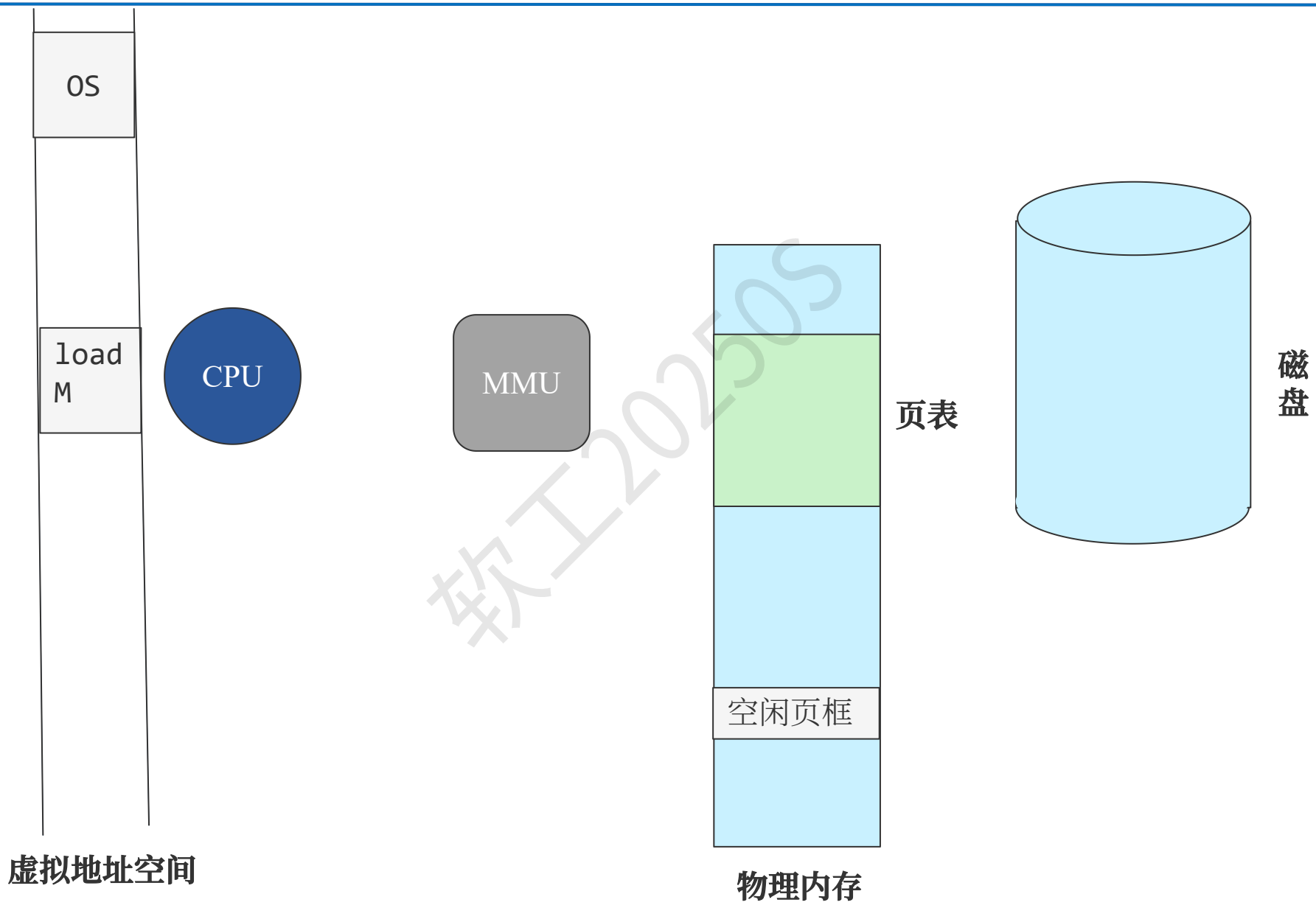
B: 某时刻触发缺页中断的地址

C: 内核中某条指令的地址

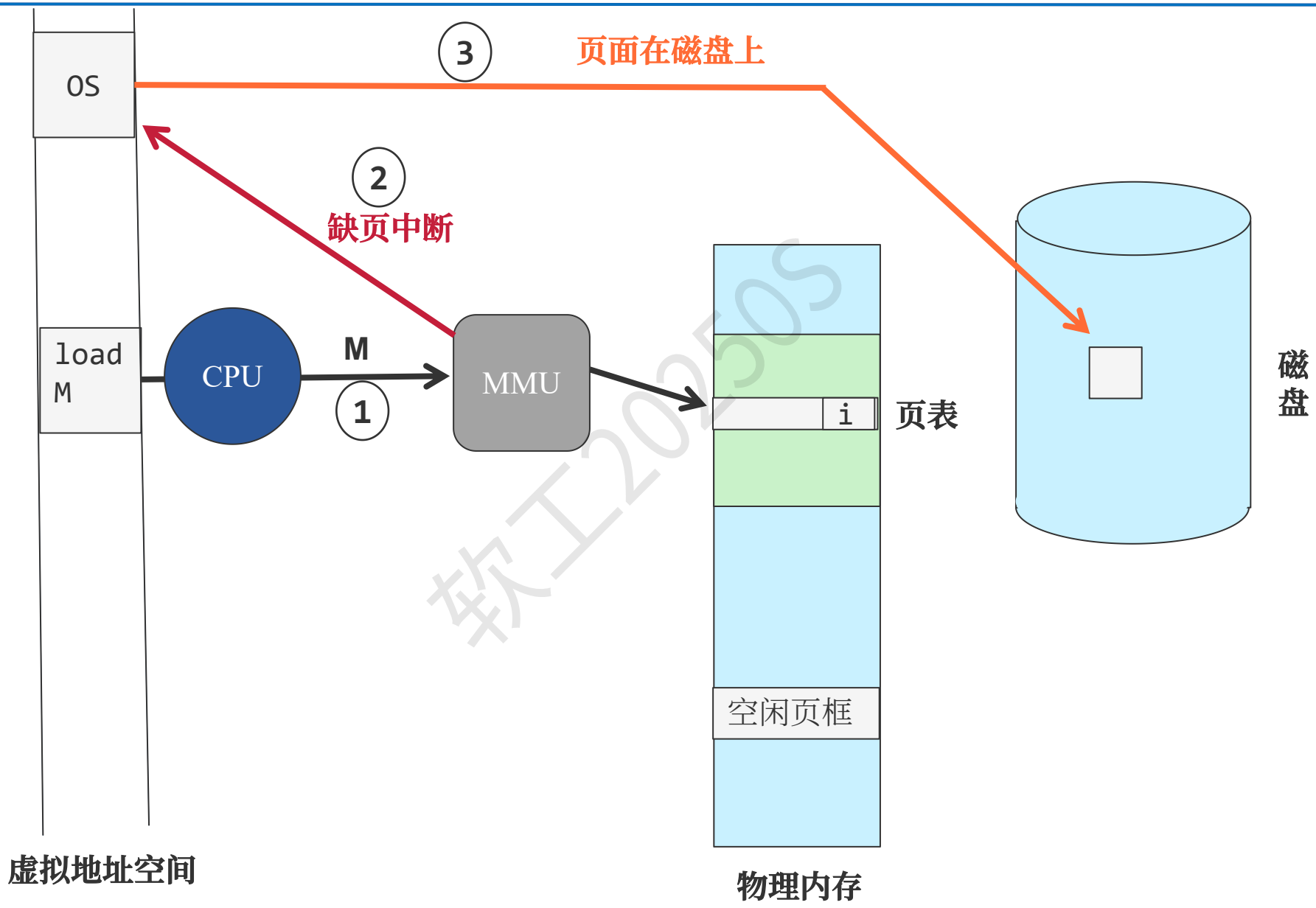
```
  switch(tf->trapno){
8010606c:       83 e8 20                sub    $0x
8010606f:       83 f8 1f                cmp    $0x
80106072:       77 08                   ja     801
80106074:       3e ff 24 85 bc 80 10    notrack jm
8010607b:       80
    lapiceoi();
    break;

  //PAGEBREAK: 13
  default:
    if(myproc() == 0 || (tf->cs&3) == 0){
8010607c:       e8 ef d8 ff ff          call   801
80106081:       85 c0                   test   %ea
80106083:       0f 84 4d 02 00 00       je     801
80106089:       8b 7b 38                mov    0x3
8010608c:       89 fe                   mov    %ed
8010608e:       f6 43 3c 03             testb  $0x
80106092:       0f 84 41 02 00 00       je     801
            tf->trapno, cpuid(), tf->eip, rcr2()
      panic("trap");
    }
    // In user space, assume process misbehaved.
    //
```

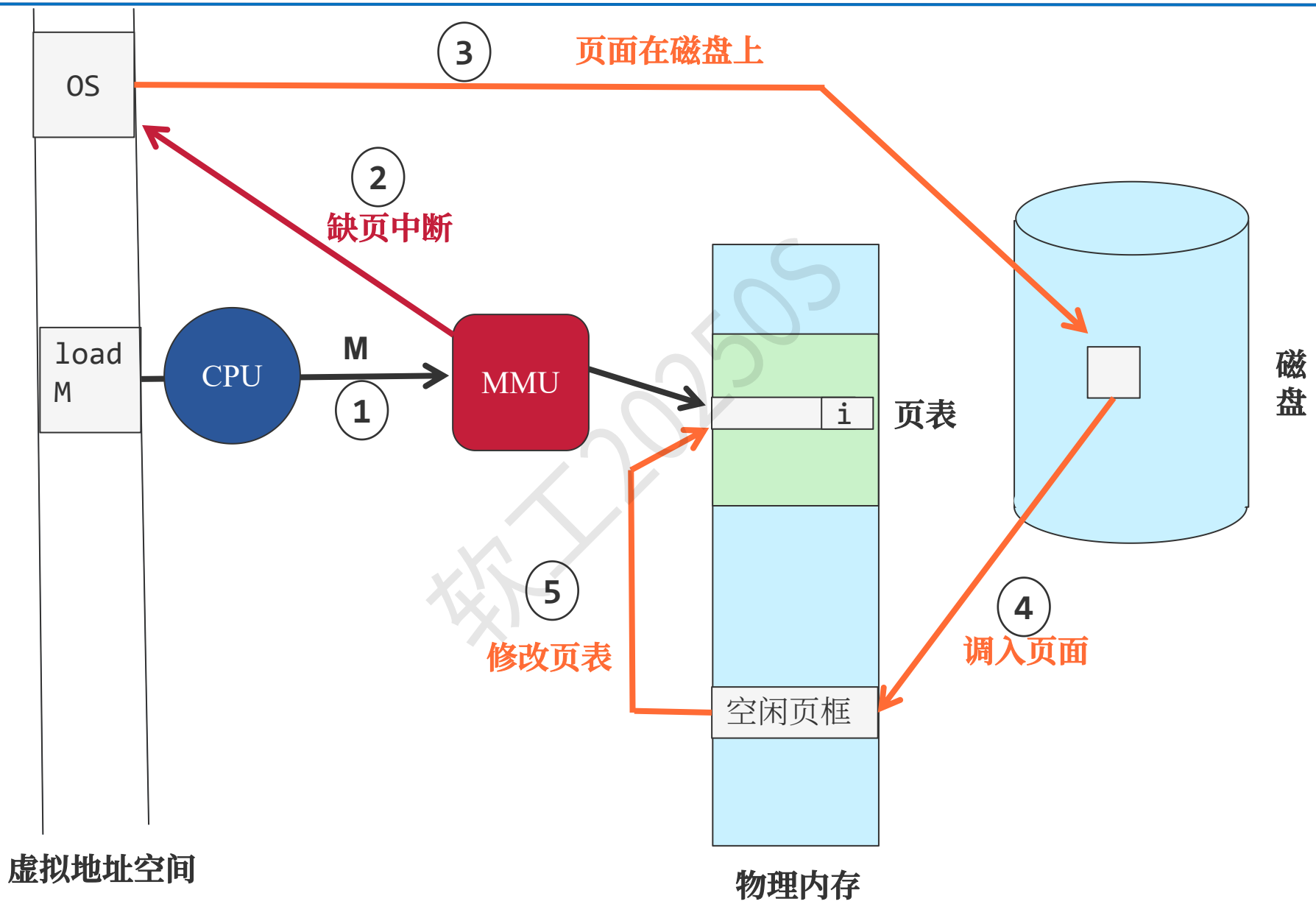
缺页中断的动态过程



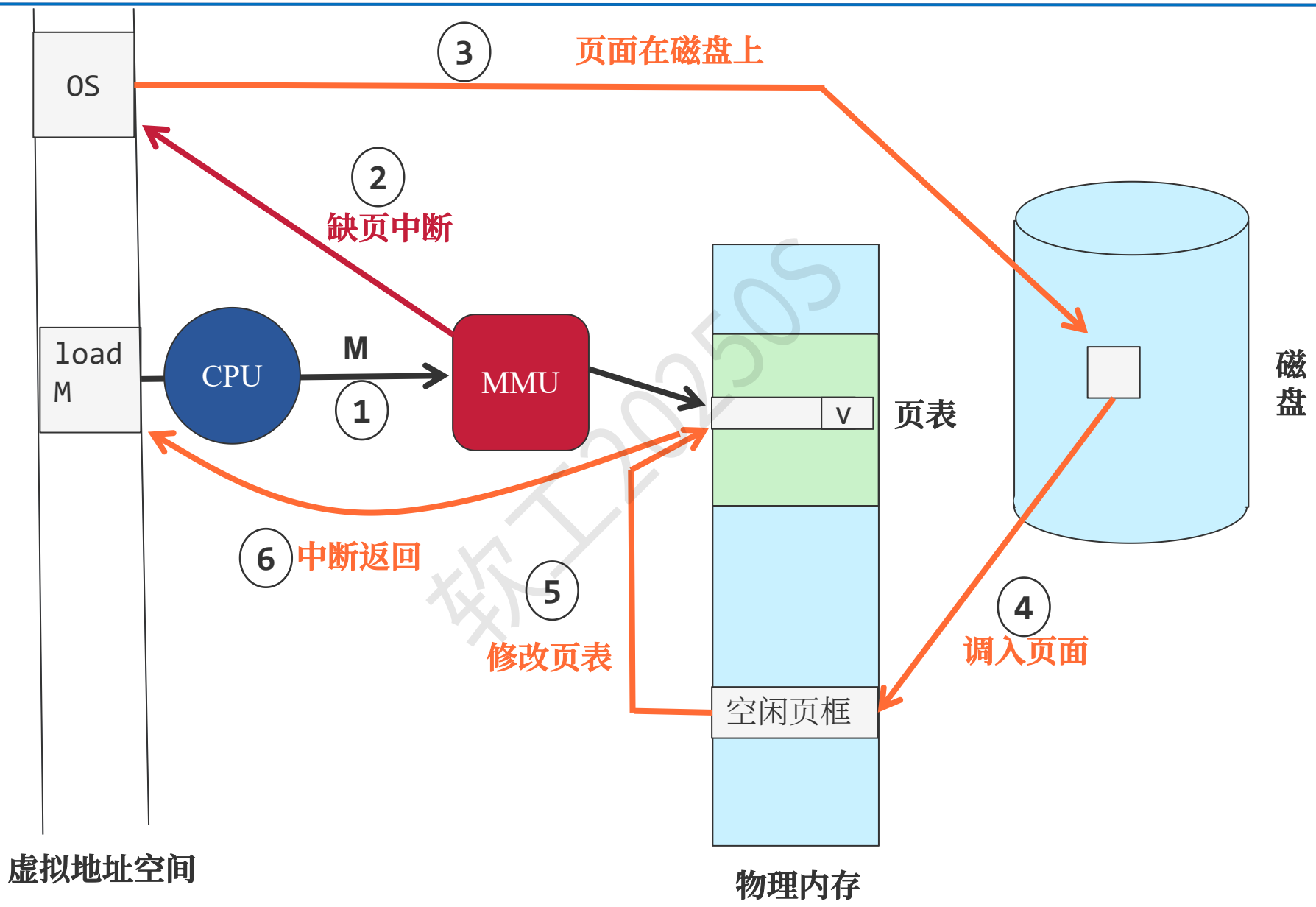
缺页中断的动态过程



缺页中断的动态过程


OS
3
页面在磁盘上
2
缺页中断
load
M
CPU
M
1
MMU
i
页表
磁盘
5
修改页表
4
调入页面
空闲页框
虚拟地址空间
物理内存

缺页中断的动态过程

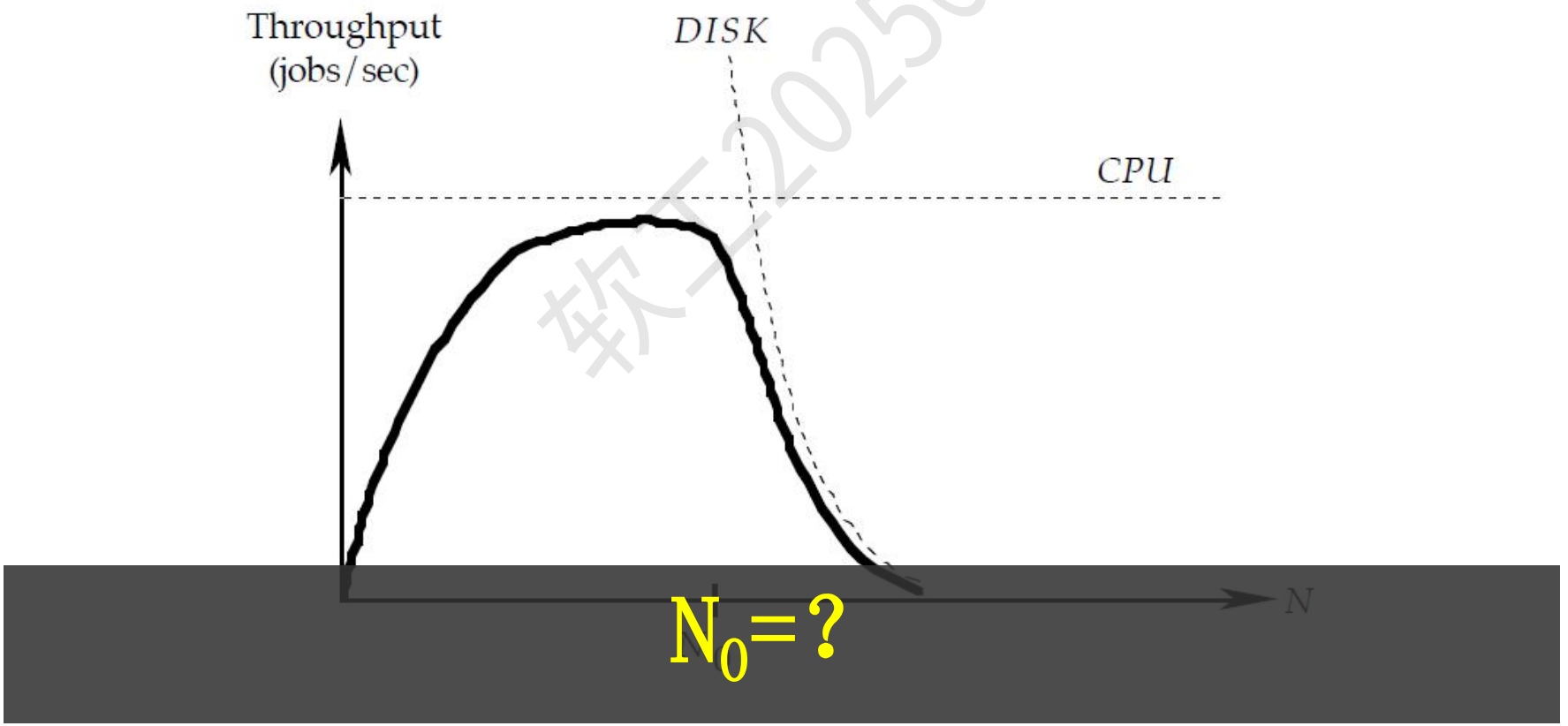

OS
3
页面在磁盘上
2
缺页中断
load
M
CPU
M
1
MMU
V
页表
磁盘
6
中断返回
5
修改页表
4
调入页面
空闲页框
虚拟地址空间
物理内存

如果没有空闲页框，一个新进程到来，OS会怎么做?

前面描述的请求分页管理中有“拒绝”机制吗?

工作集算法：起因

- 抖动（thrashing）
  - 调到外存的页面很快又被访问，又需要调入内存
- 1960s 所观察到的现象



工作集算法

- 工作集
  - 1968年由 Denning 提出，用于解决thrashing
  - 进程在一段时间（以自己的时间为准）delta内访问的页面的集合
  - w(t,delta) 包含于 w(t,delta+1)
- 算法思想
  - 将不在工作集中的页面替换出去
  - 如果都在工作集中怎么办？**拒绝！**
    - 暂停一些进程（说明进程太多）

页面置换算法

- 最佳置换算法
- 先进先出置换算法（FIFO）
- 最近最久未使用置换算法（LRU）
- Clock置换算法
- 改进型Clock算法
- 工作集算法

练习

- 单级页表，CPU在执行指令 mov eax,[eax] 时，可能触发几次缺页中断?

- 取指令本身可能触发几次?
  - 0次，如果所在的页面已经在内存中
  - 1次，如果指令在某一页上
  - 2次，如果指令跨页
- 写内存[eax]可能触发几次?
  - 0次，如果所在的页面已经在内存中
  - 1次，如果不在，但在一页上
  - 2次，如果不在，并且跨页

0-4次

虚拟内存管理要求（2025考研大纲）

- 虚拟内存的基本概念
- 请求页式管理
- 页置换算法

写时复制

- shell如何工作?
  - fork，在子进程中 exec
  - fork出一个地址空间，分配物理内存，接着执行exec，创建新的地址空间，分配物理内存，最后将fork中分配的物理内存归还
- 写时复制的fork
  - 将子进程与父进程指向同样的页框
  - 标记可写页面为只读，这些页面为写时复制页面（利用页表项的空闲位）
  - 当试图写写时复制页面时触发缺页中断，再分配