反向映射RMAP系统简介

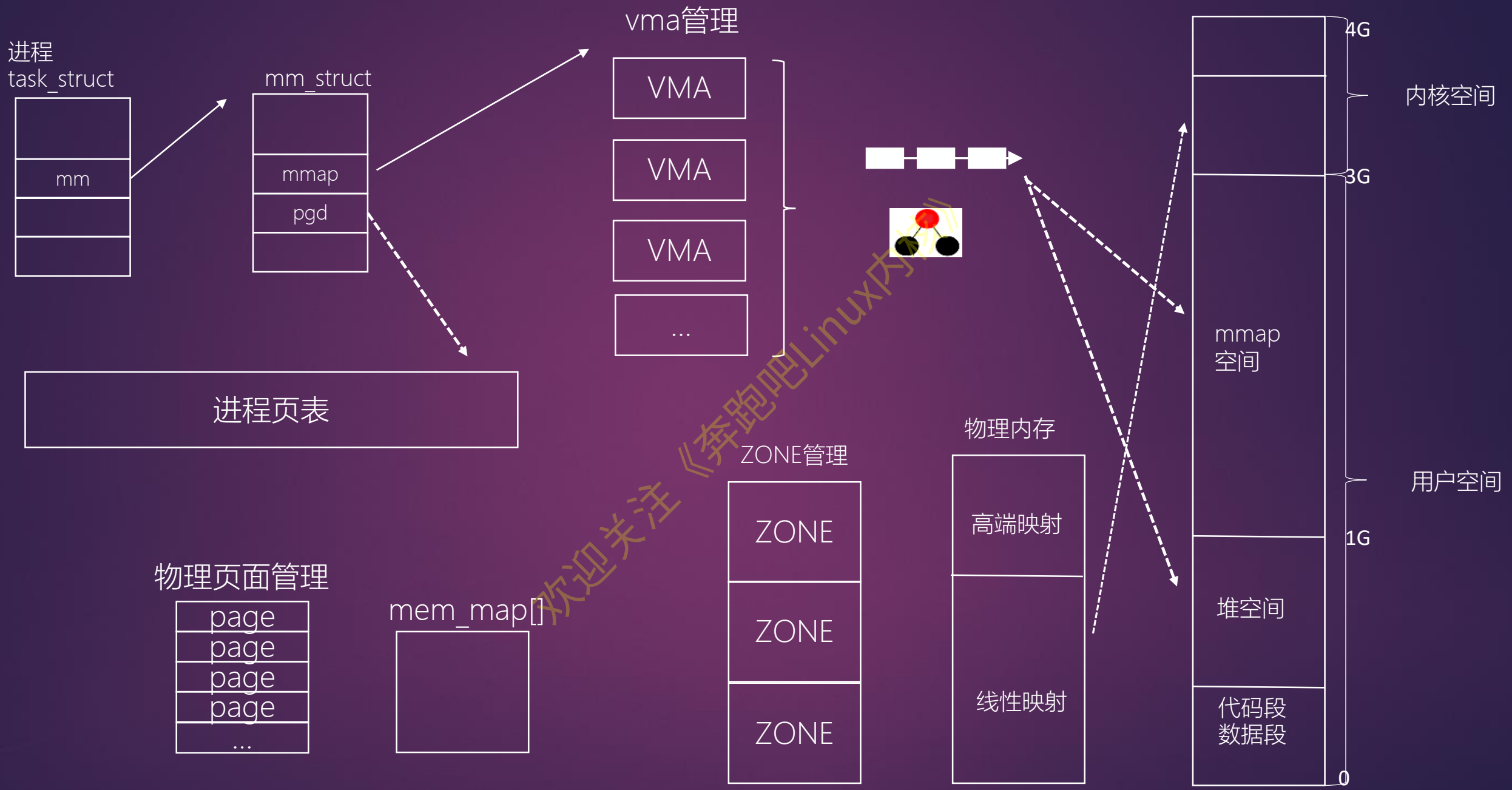
笨叔叔

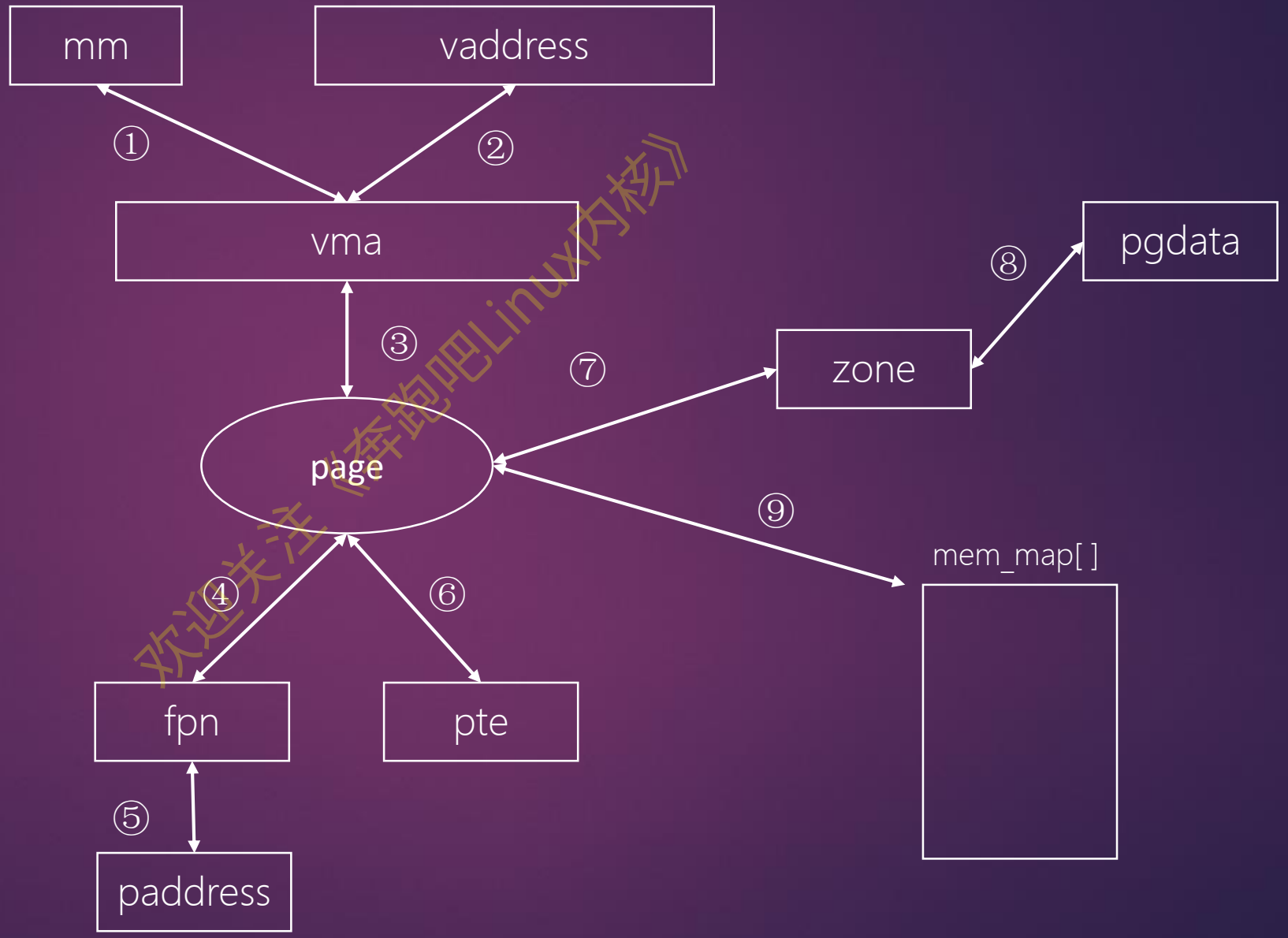
Agenda

- 内存管理总览
- 反向映射来由
- 图解Linux 2.6.11的RMAP
- 图解Linux 4.0的RMAP

内存管理总览


用户进程
用户进程
用户进程
...
malloc/mmap/mlock/madvise/mremap/...
用户空间
sys_brk
sys_mmap
sys_madvise
sys_...
vma管理
缺页中断
匿名页面
page cache
页面回收
slab
反向映射
KSM
Huge Page
页迁移
内存规整
内核空间
伙伴系统（页面分配器）
页表管理（内核页表和进程页表）
MMU
TLB
cache
物理内存
硬件层
欢迎关注《奔跑吧Linux内核》



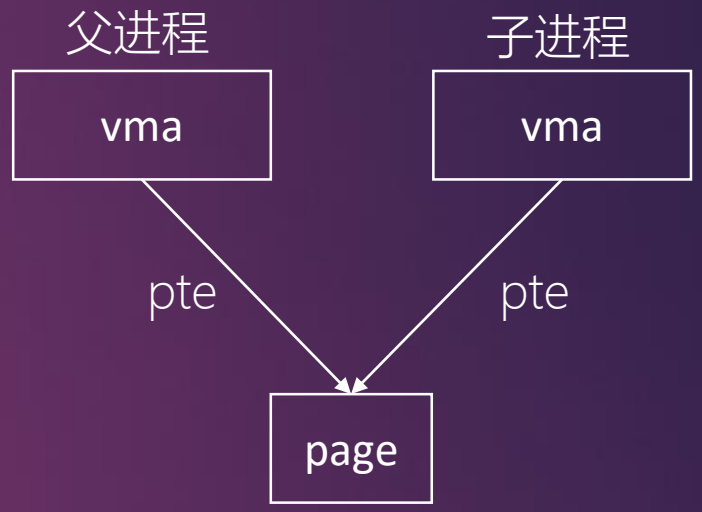


反向映射产生的来由

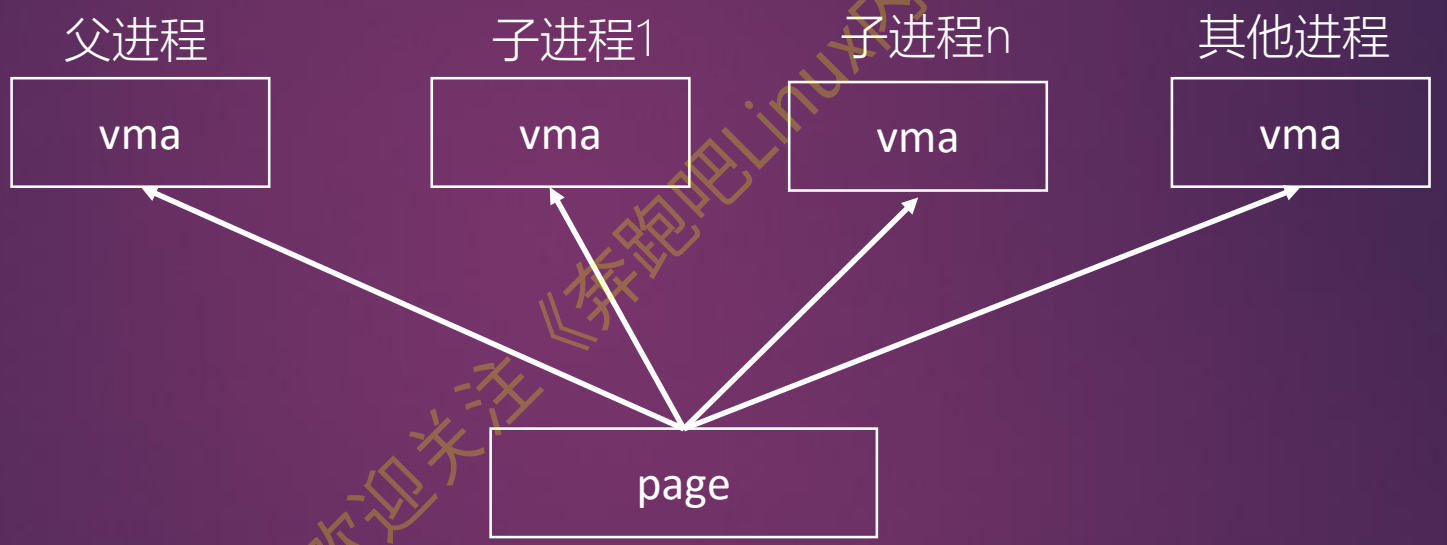
- 当物理内存短缺时？
  - ✓虚拟内存常常大于物理内存
  - ✓把暂时不用的物理内存swap到交换分区

- 如何判断哪些物理内存暂时不用？
  - ✓LRU算法
  - ✓第二次机会法

- Linux 2.4内核的做法
  - ✓遍历所有进程的VMA来确定某个物理页面映射的pte
  - ✓耗时，低效



反向映射的设计目标

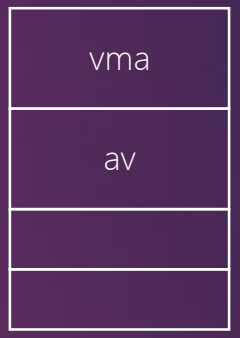


反向映射用到的数据结构

struct vm_area_struct

struct anon_vma

struct anon_vma_chain



struct page

RMAP四部曲（以Linux 2.6.11为例）

1. 父进程分配匿名页面
   - do_anonymous_page()->page_add_anon_rmap()
   - page_add_anon_rmap()
     - ✓page->mapping指向VMA的anon_vma数据结构
     - ✓计算page->index

2. 父进程创建子进程
   - do_fork()->copy_mm()->dump_mm(), 把父进程所有的VMA复制到子进程对应的VMA中
   - anon_vma_link(): 把子进程的VMA加入到父进程的vma->anon_vma->head链表中

3. 子进程发生COW

- 当父子进程共享匿名页面，子进程的VMA发生COW

-> 缺页中断

->handle_pte_fault()

->do_wp_page()

->分配一个新的匿名页面

->page_add_anon_rmap()

新分配的匿名页面page->mapping指向父进程vma->anon_vma

- 4. RMAP应用
  - ➢页面回收：断开所有映射的用户PTE才能回收页面
  - ➢页面迁移：断开所有映射的用户PTE

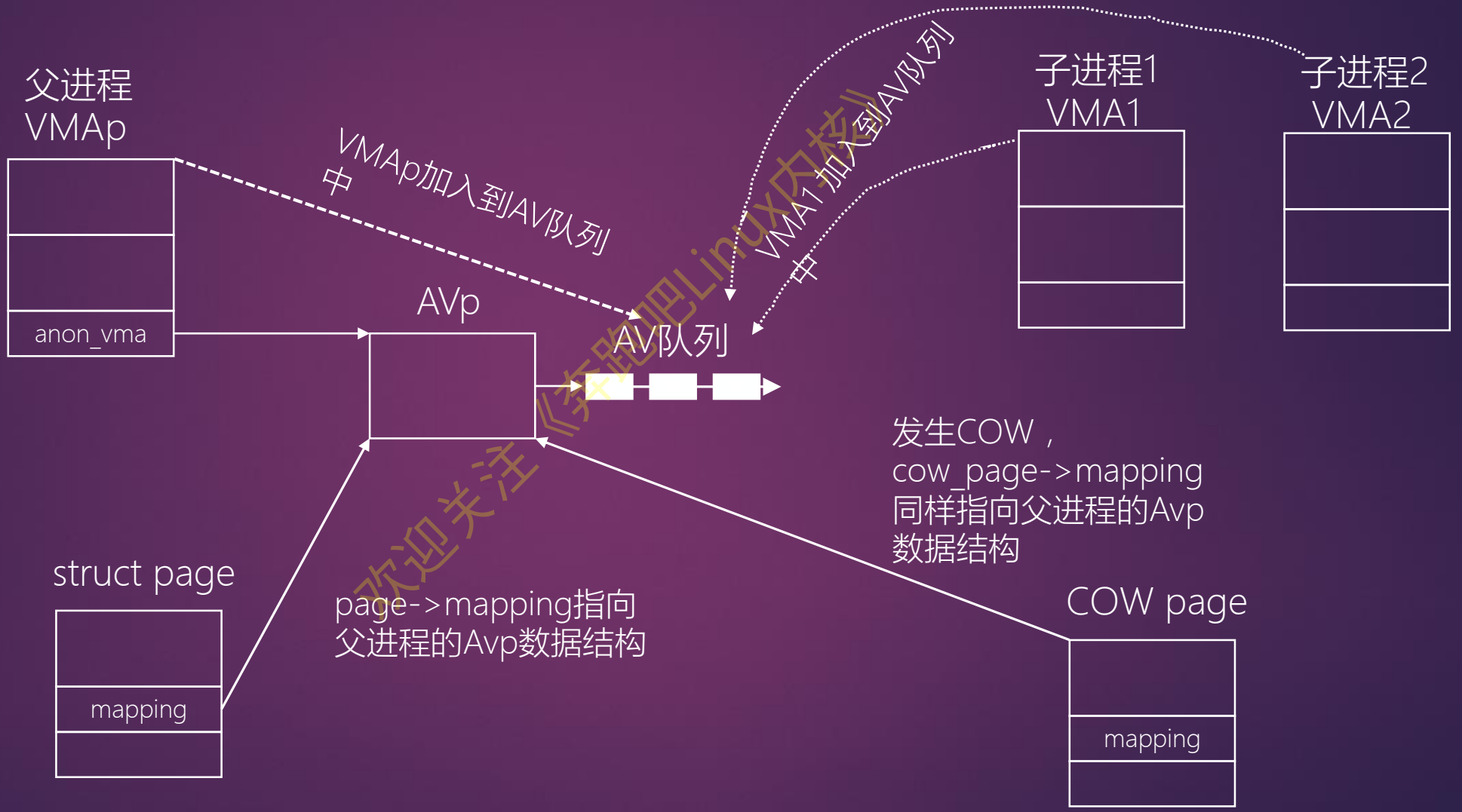
- ➢try_to_unmap()
  - ✓try_to_unmap_anon()
  - ✓try_to_unmap_file()

```
static int try_to_unmap_anon(struct page *page)
{
        anon_vma = page_lock_anon_vma(page);

        list_for_each_entry(vma, &anon_vma->head, anon_vma_node) {
                ret = try_to_unmap_one(page, vma);
        }
        return ret;
}

static int try_to_unmap_one(struct page *page, struct vm_area_struct *vma)
{
        address = vma_address(page, vma);
        …
}
```

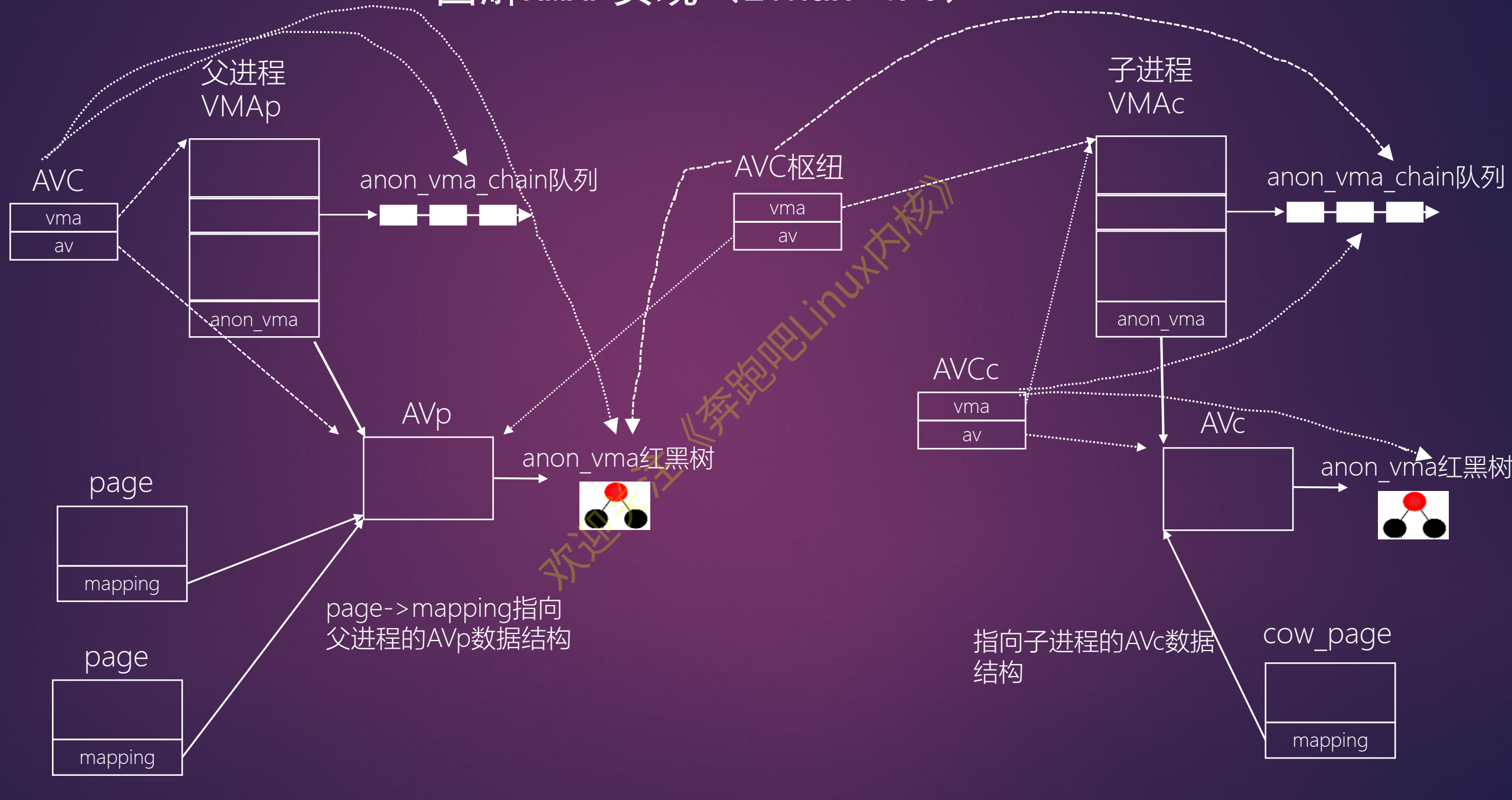
图解RMAP实现（Linux 2.6.11）



面临的问题和挑战

- 如果父进程有1000个子进程，每个子进程都有一个VMA，这个VMA里面有1000个匿名页面，当所有的子进程的VMA同时发生写时复制时会是什么情况呢？

图解RMAP实现（Linux 4.0）



总结

- 提高页面回收效率
- 消耗一定内存空间
- 连接虚拟内存管理和物理内存管理的一个桥梁

- 如果多个VMA的虚拟页面同时映射了同一个匿名页面，那么此时page->index应该等于多少？