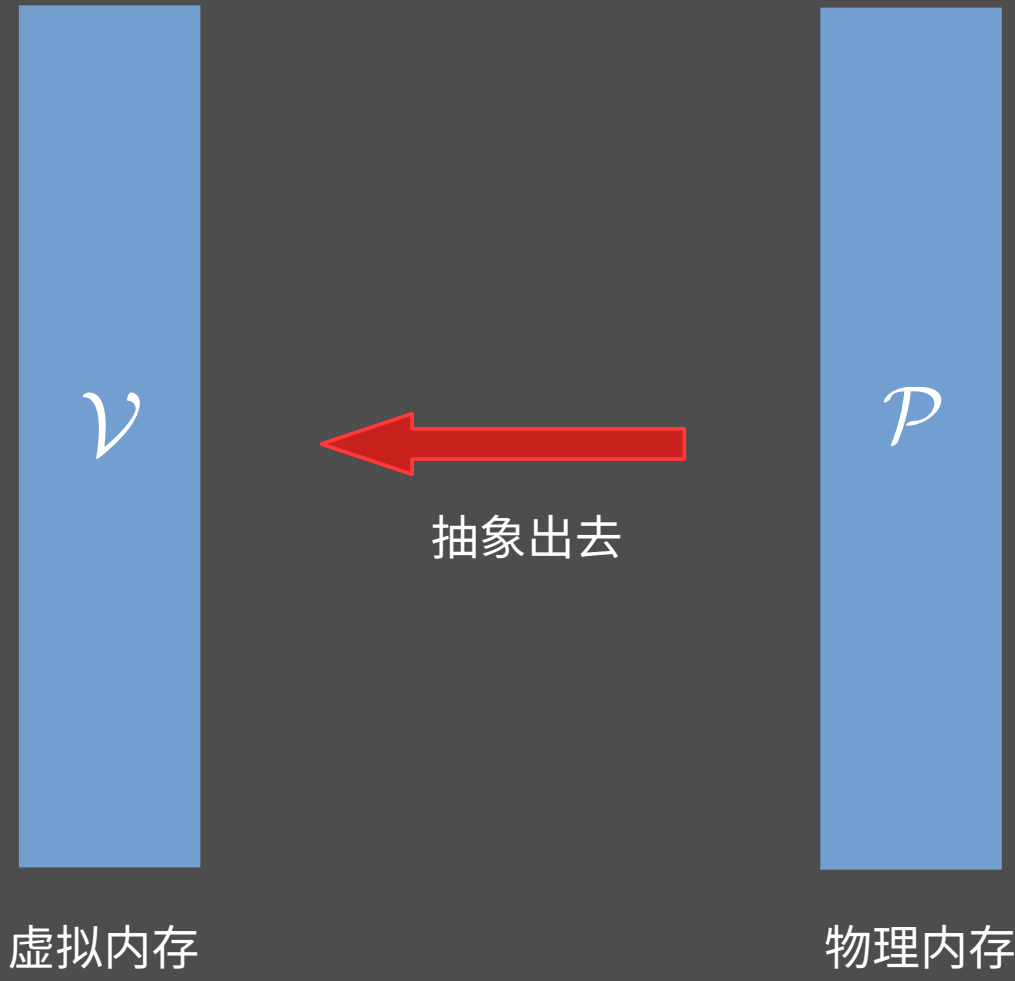
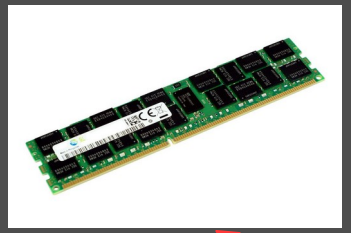
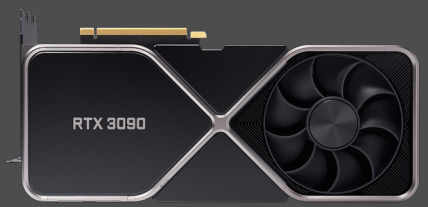

LUNIX
Operating System
从零开始
DEVELOPING YOUR OWN
自制操作系统
OPERATING SYSTEM FROM SCRATCH
O Brave New Kernel
虚拟内存：分页与内核重映射
OPERATING SYSTEM DEVELOPMENT
TUTORIAL SERIES
EP 7-1

何为虚拟内存?

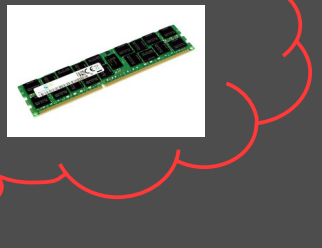


一个“标准化”的问题





虚拟内存



虚拟内存 - CPU 的幻觉

不同的存储介质有不同的地址空间，而 CPU 希望只处理一个统一的地址空间

如何将需要将不同的地址空间映射至一个统一的地址空间?

Well... 就映射嘛!

物理地址空间 $\mathcal{P} = \{0, 1, 2, \ldots, N_p\}$

虚拟地址空间 $\mathcal{V} = \{0, 1, 2, \ldots, N_v\}$

地址翻译:

$$F : \mathcal{V} \mapsto \mathcal{P} \cup \{\varnothing\}$$

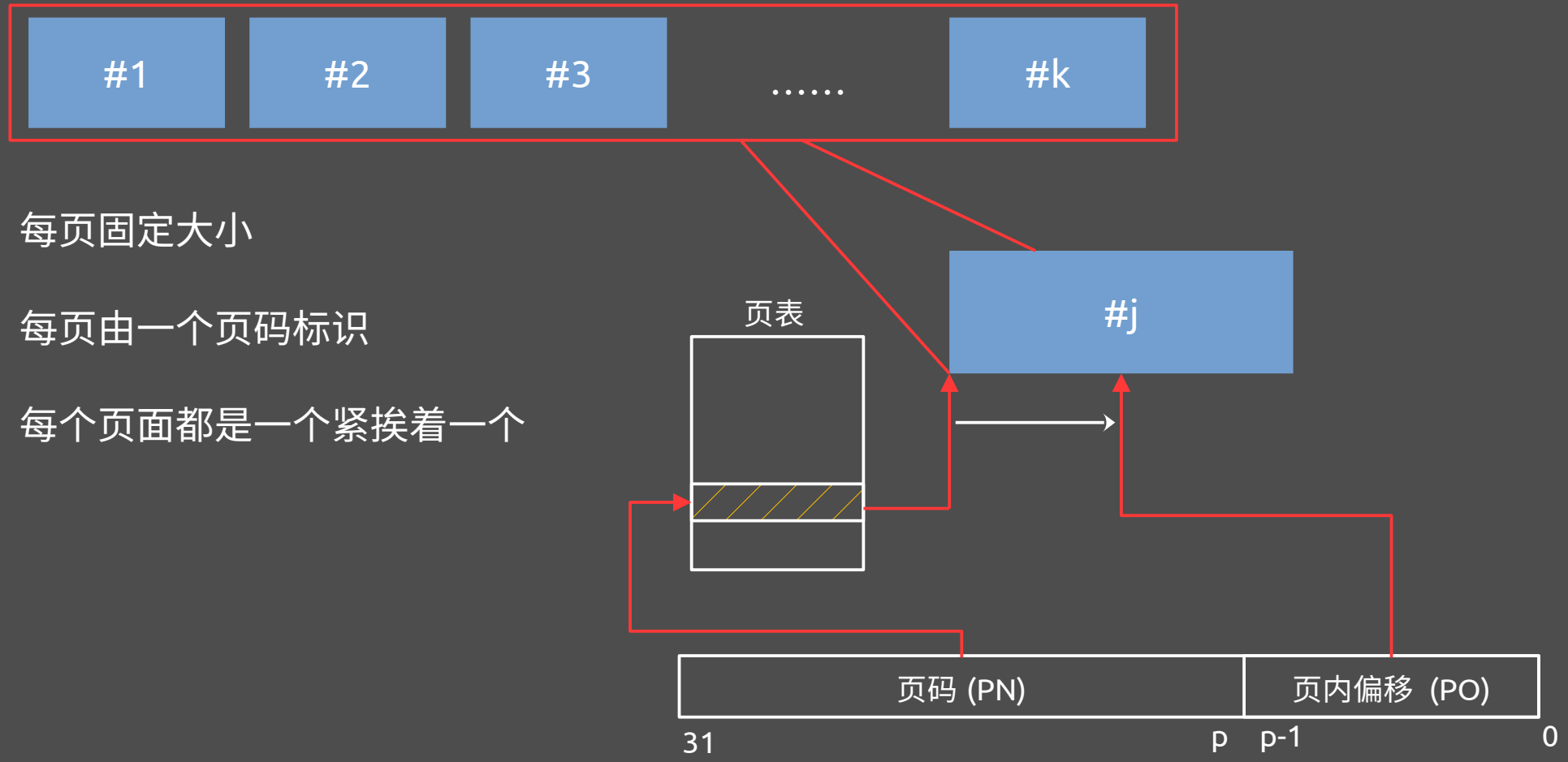
如何映射?

一个极复杂的数学表达式?

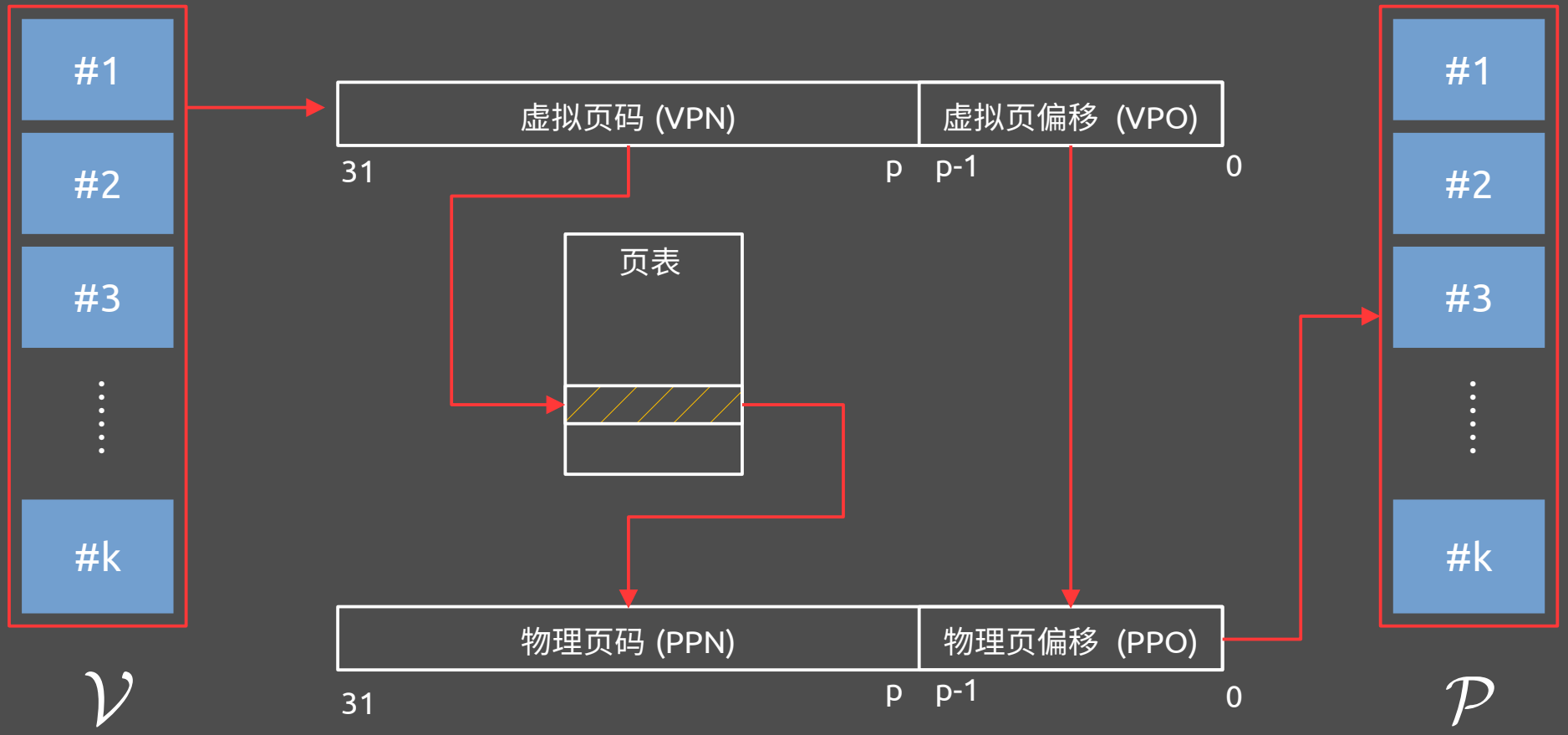
一个巨大的 Hash table ?

分页

虚拟内存 – 分页 （ Paging ）



$$F : \mathcal{V} \mapsto \mathcal{P} \cup \{\varnothing\}$$



这样的效率很高吗?

不需要存储每个字节地址的映射关系!

直接在现有的地址上操作

维持了地址的连续性 —— 不需要操心 Corner case

时间复杂度: O(1) 的算法!　　P_{addr} = pageTable[PN >> p] + PO

空间复杂度?

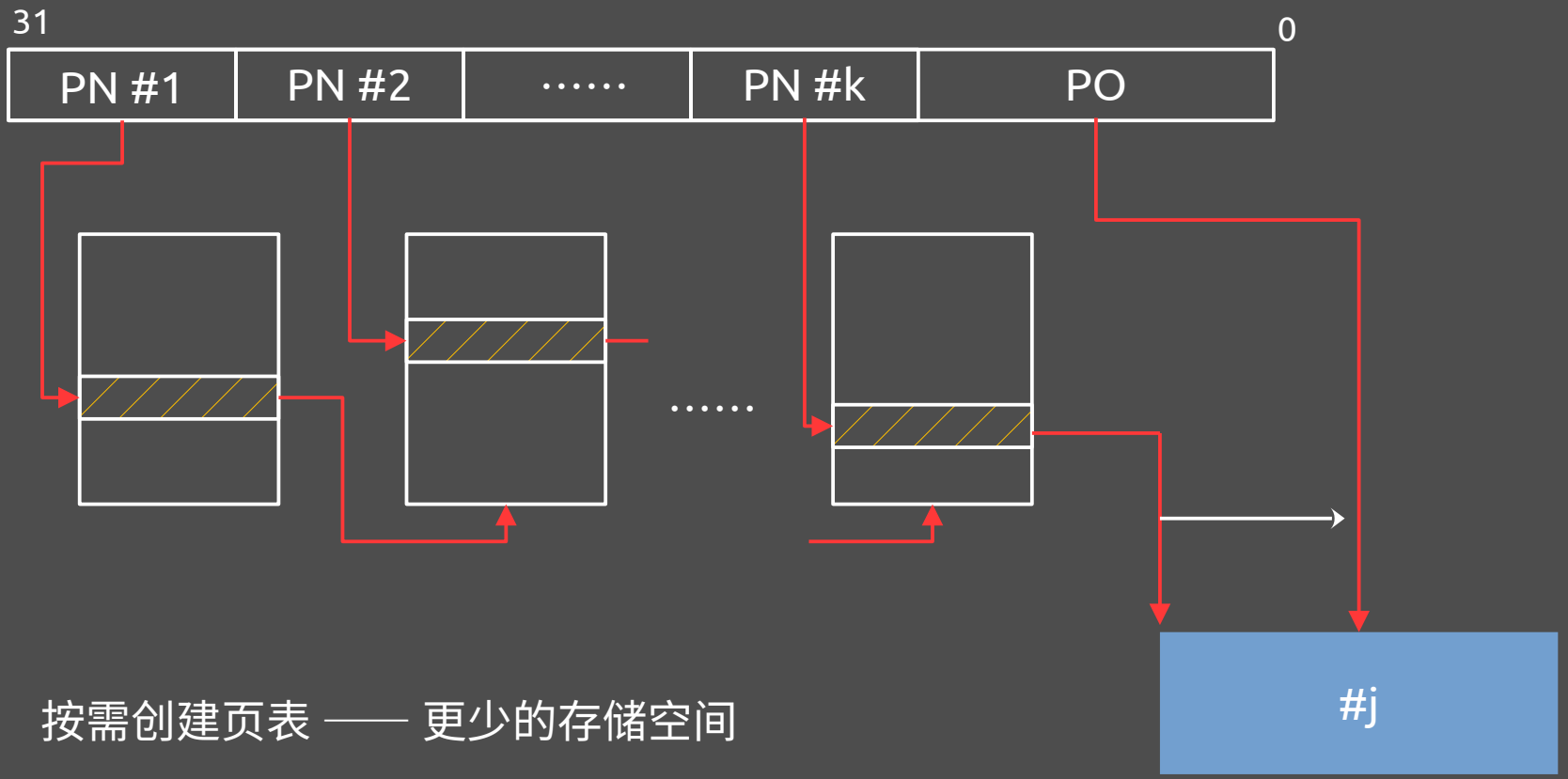
页表 pageTable 的大小（假设 4KiB 页大小）:

4GiB / 4KiB = 2^{20} 个元素

即 4MiB 的大小!

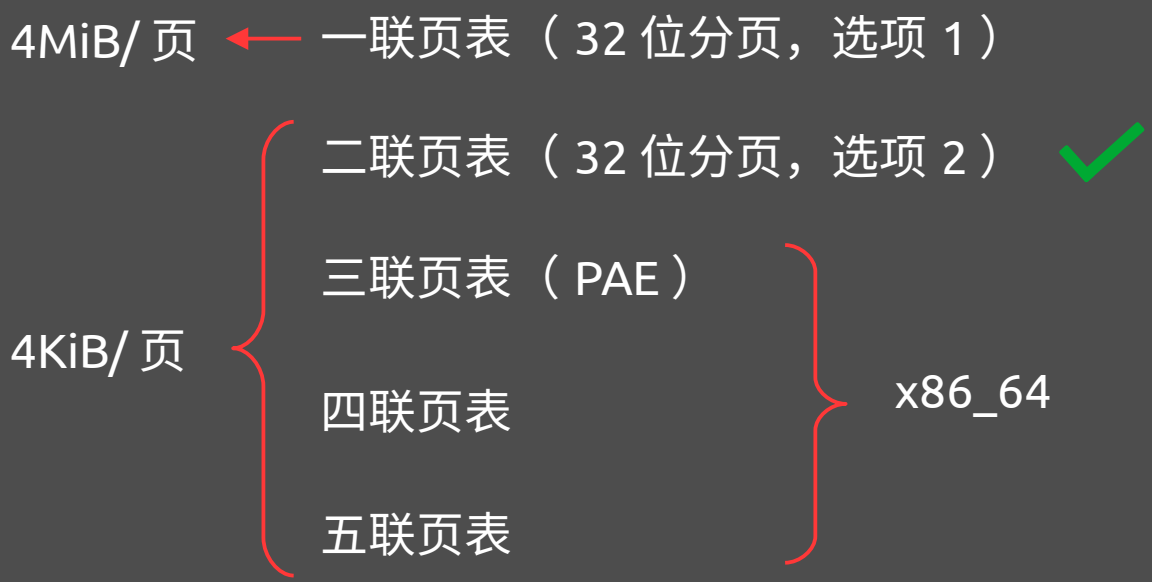
继续优化?

联级页表（Multi-level Page Table）

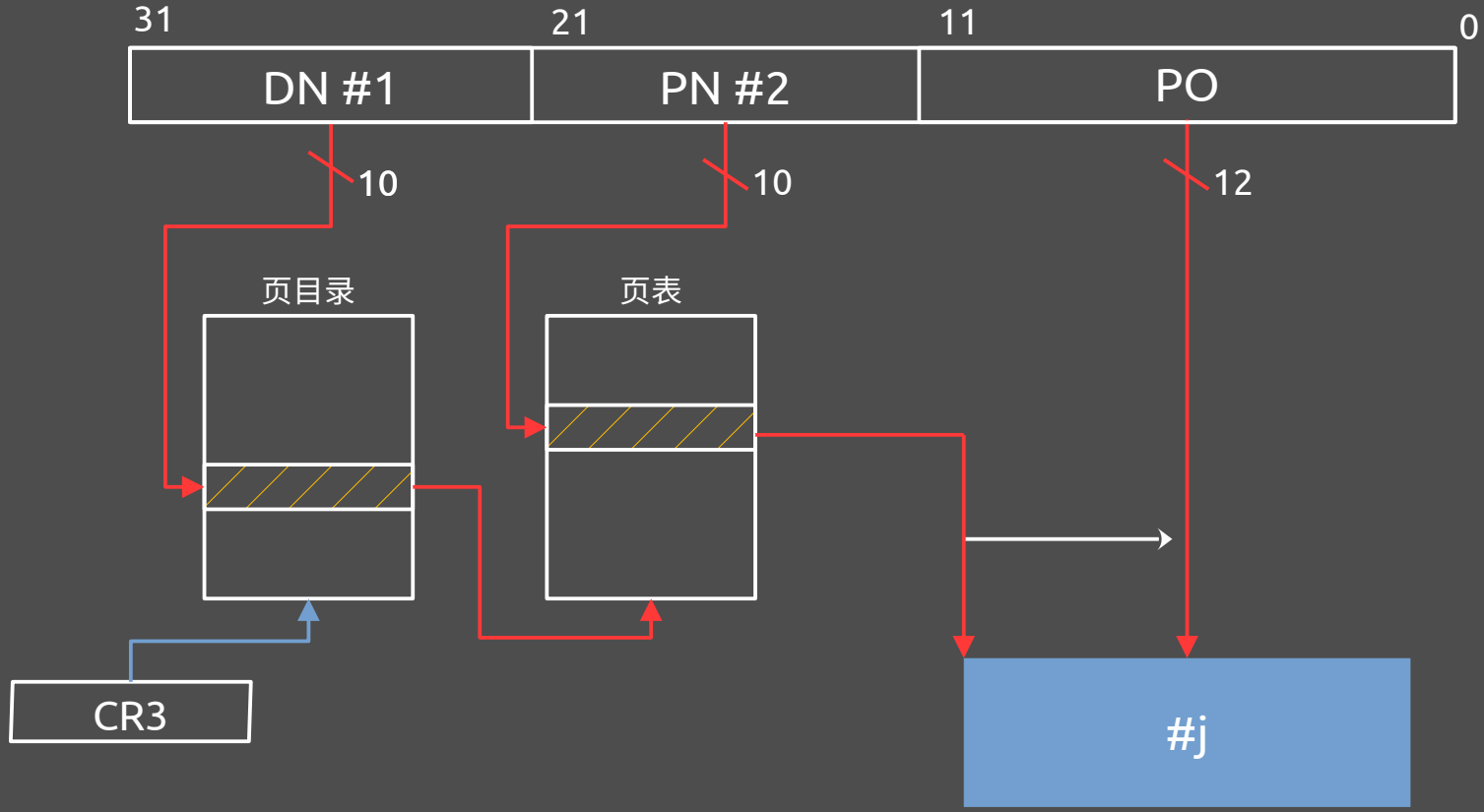


按需创建页表 —— 更少的存储空间

x86 下的分页机制


4MiB/ 页
一联页表（ 32 位分页，选项 1）
4KiB/ 页
二联页表（ 32 位分页，选项 2）
三联页表（ PAE）
四联页表
五联页表
x86_64

32 位分页 - 4KiB 每页



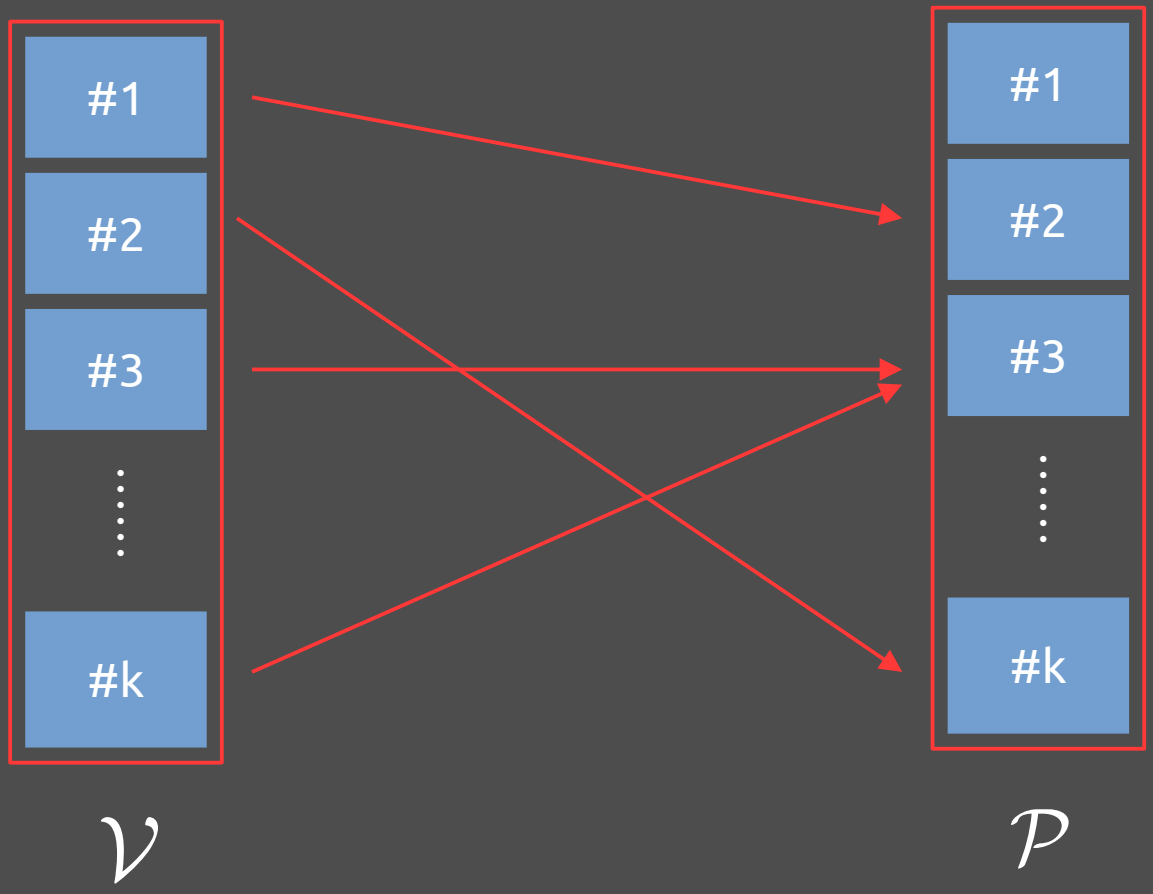
页表与页目录最多包含 1024 个项目

每页大小 4KiB ， 起始地址 4Ki 对齐

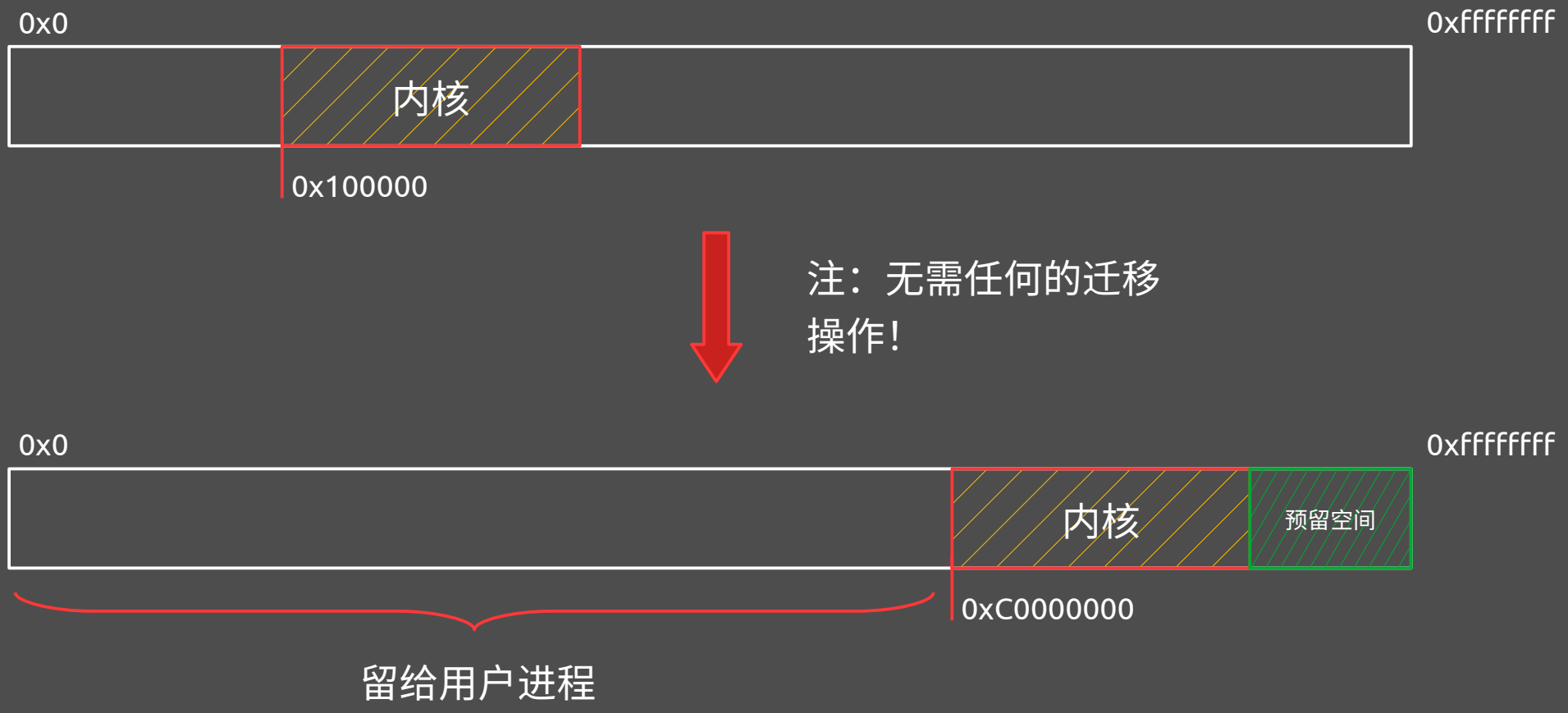
虚拟内存带来的机遇

允许更加简单高效的内存管理	→	最小管理单位是固定大小的页，而不是字节或者是可变长度的段！
允许我们自定义内存布局	→	页表任由我们填写
简化任何文件读取或者程序加载的操作	→	比如，映射磁盘中的某个区域？
每个进程都有其专享的，私有的，内存	→	我们可以为每个进程指定不同的映射
允许共享内存	→	映射进同一块儿物理页就行了

内存管理 – 不需要各就各位



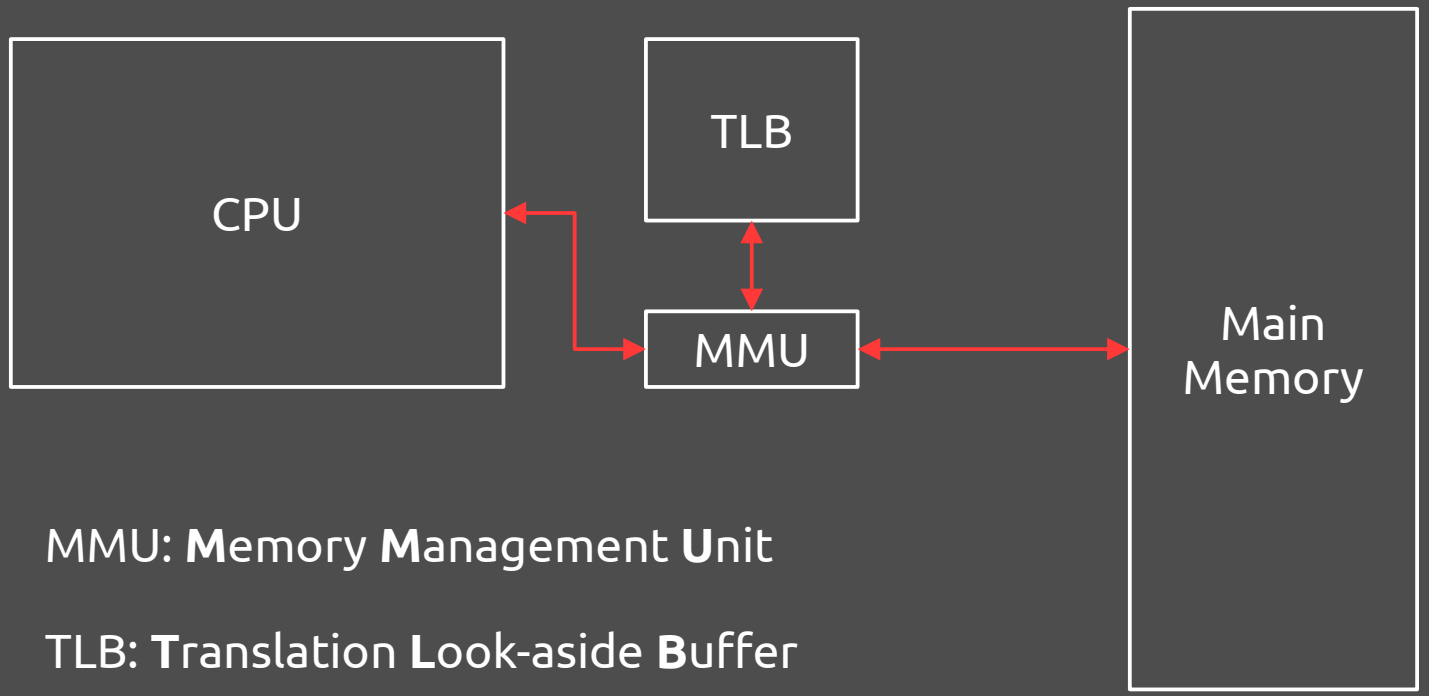
自定义内存布局 - 内核重映射



Higher-half Kernel （高半核？）

虚拟内存带来的挑战

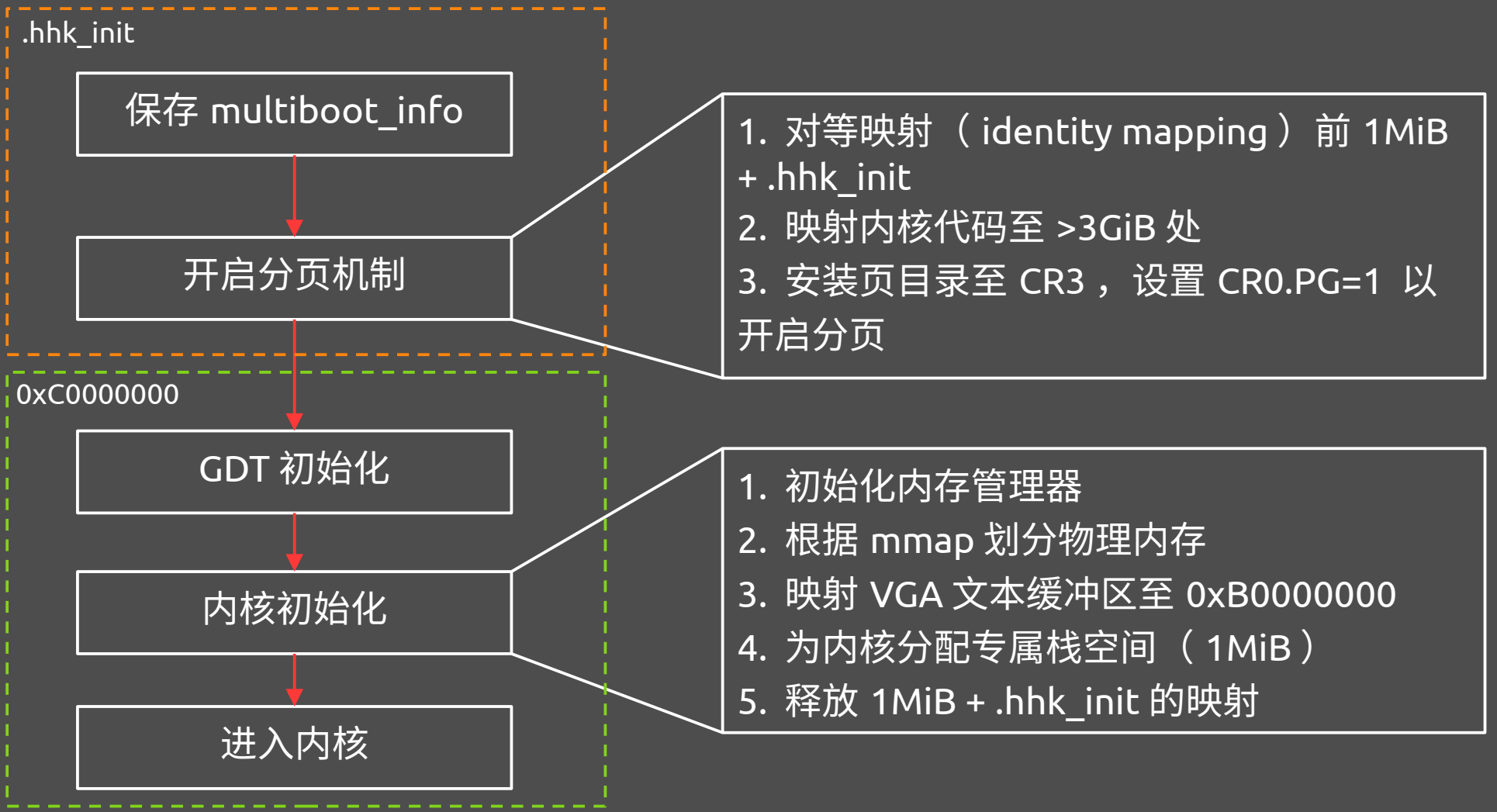
一切皆虚拟



MMU: **M**emory **M**anagement **U**nit

TLB: **T**ranslation **L**ook-aside **B**uffer

LunaixOS 内核加载流程



Code Time!


LUNIX
Operating System
从零开始
DEVELOPING YOUR OWN
自制操作系统
OPERATING SYSTEM FROM SCRATCH
O Brave New Kernel
虚拟内存：内存管理
OPERATING SYSTEM DEVELOPMENT
TUTORIAL SERIES
EP 7-2

内存管理 - 设计目标

物理内存管理：

- 分配可用页（用于映射）
- 对合适的页进行 Swap 操作（暂时不需要）

虚拟内存管理：

- 管理映射 - 增删改查（ CRUD ）
  - 增： VA ⟷ PA
  - 删：删除映射（删除对应表项，释放占用的物理页，刷新 TLB ）
  - 改：修改映射
  - 查： VA ⟶ PA

物理内存管理

需要记录所有的物理页的可用性

4GiB 大小 = 2^{20} 个记录！

简单实现：位图法（ Bitmap ）

每一个 bit 代表一个物理页，

0： 可用

1： 已占用

需要 128Ki 的空间

获取页号为 PPN 的物理页可用性：

```
bitmap[PPN / 8] & (0x80U >> (PPN % 8))
```

Code Time!

虚拟内存管理

现成的存储：页表与页目录

操作直接照搬上期所讲的地址转换方法，简单！

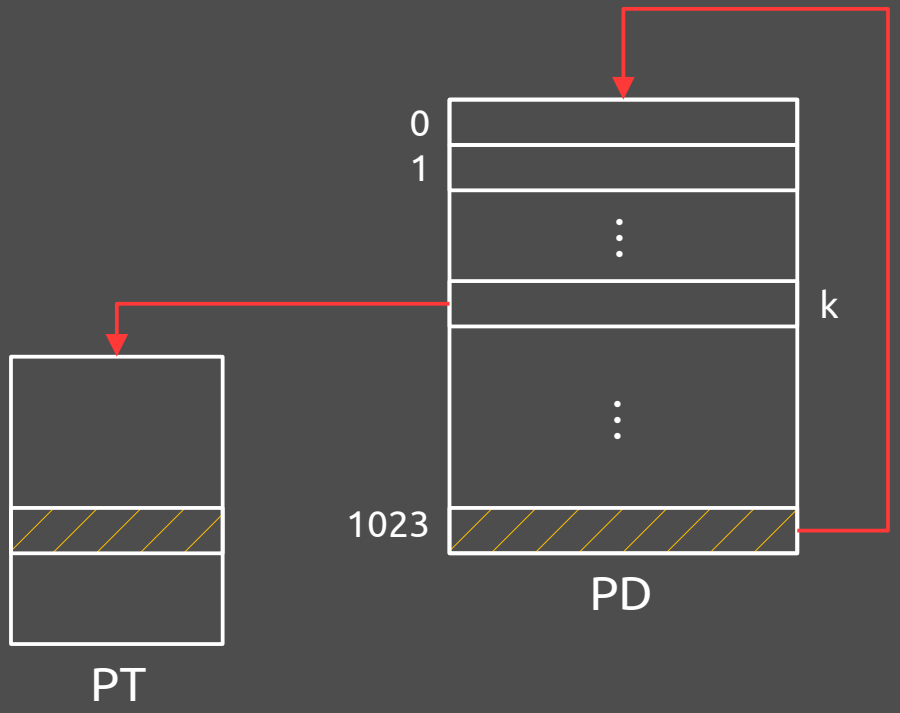
But ...

页表与页目录存放的是物理地址
我们需要一个虚拟地址的映射才可以访问他们
我们需要访问页表与页目录才能够建立映射
但我们需要虚拟地址…… F*ck!

解决方案：递归映射！

递归映射我们的页表与页目录

递归映射：自己指向自己



PD 的虚拟地址： `0xFFFFF000`

PT 的虚拟地址：

```
0xFFC00000 | (PTN << 12)
```

Code Time!