[6.虚拟内存和物理内存的管理 1](#_Toc105416395)

[延迟映射/按需调页 2](#_Toc105416396)

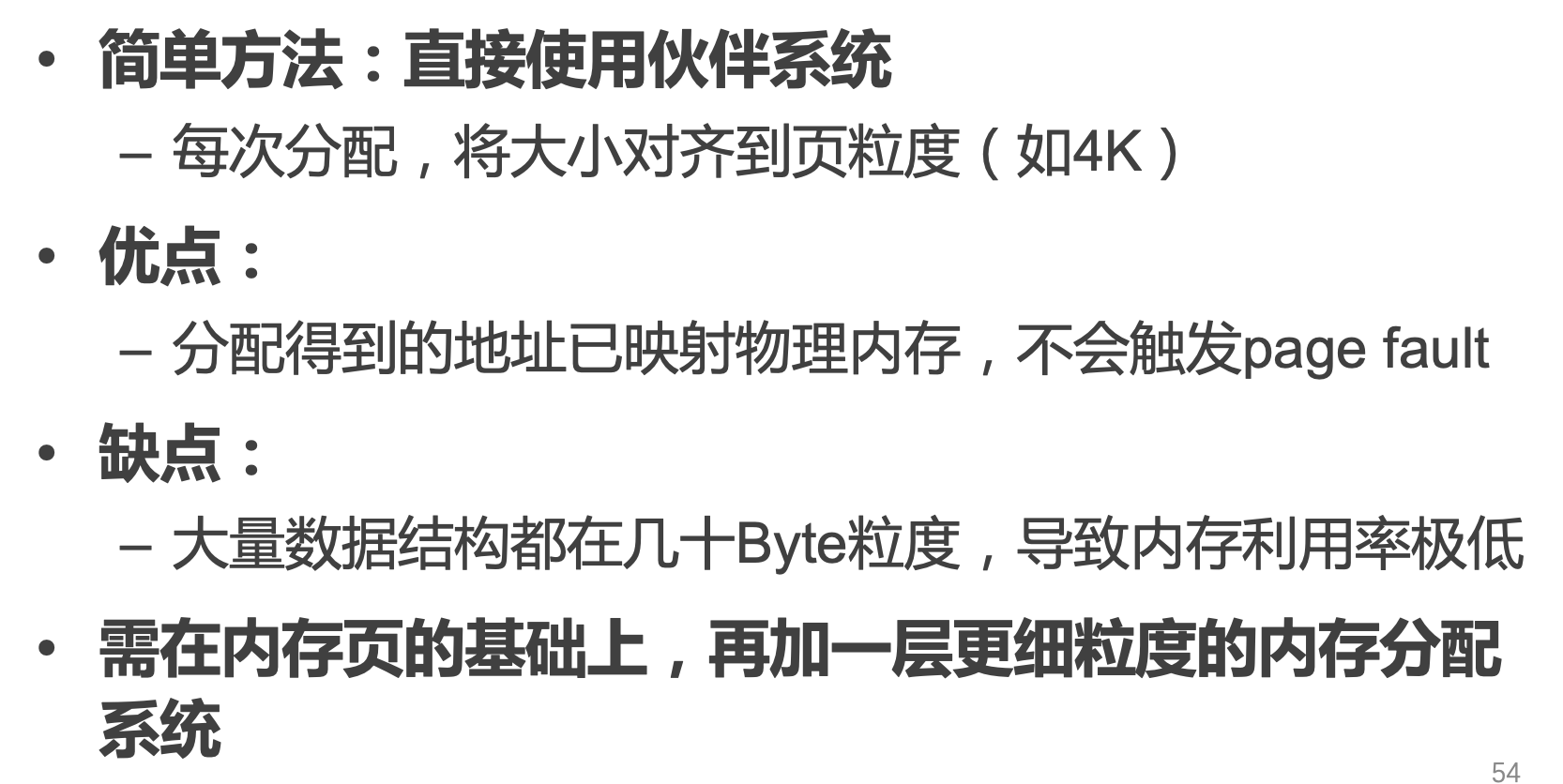
[换页机制 6](#_Toc105416397)

[OS内存管理更多机制 7](#_Toc105416398)

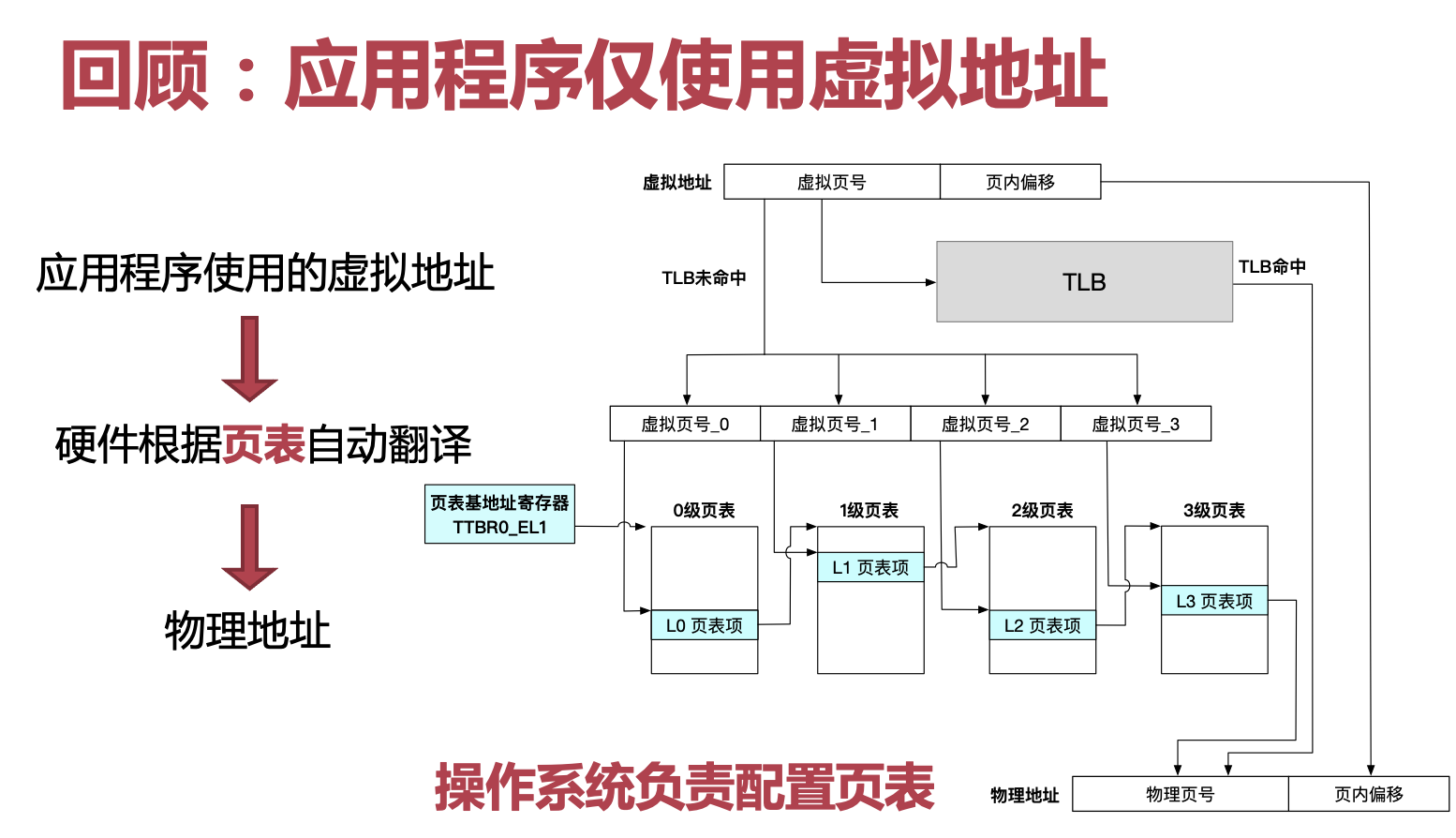
[物理内存管理 11](#_Toc105416399)

[伙伴系统：页粒度内存管理 15](#_Toc105416400)

[buddy system 15](#_Toc105416401)

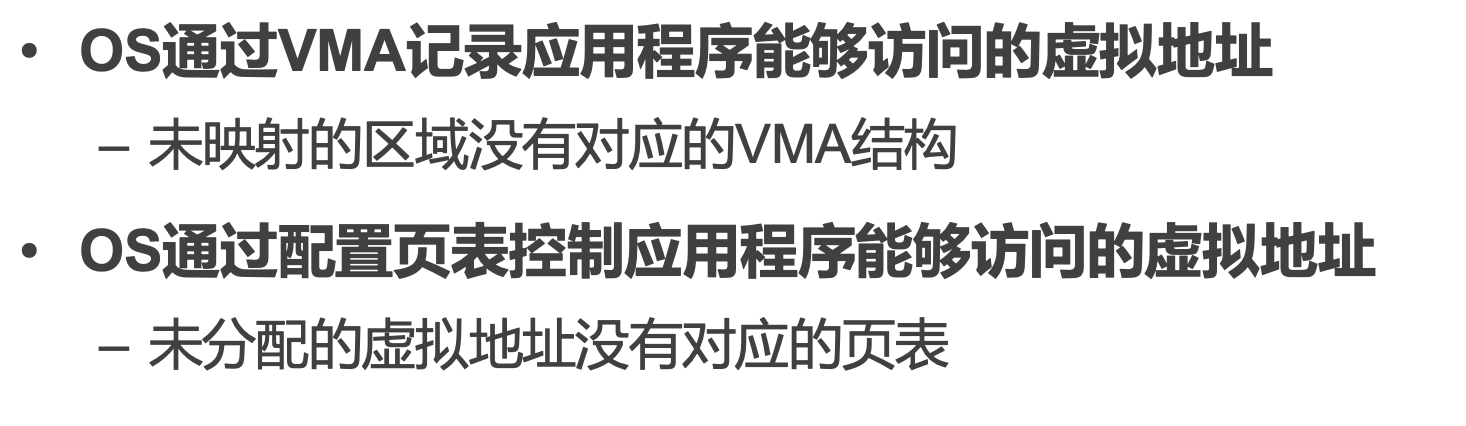
[如何实现内核的linux？  16](#_Toc105416402)

# 6.虚拟内存和物理内存的管理



页表维护了虚拟地址到物理地址的映射。

Q：VMA和页表是否冗余？



仅仅是用来判断一个应用程序是否能够访问某个虚拟地址，是否是冗余的？

可以不用VMA，因为页表记录了完整的映射，没有页表则表示没有映射。但VMA还是可以提高速度，比遍历页表更快。

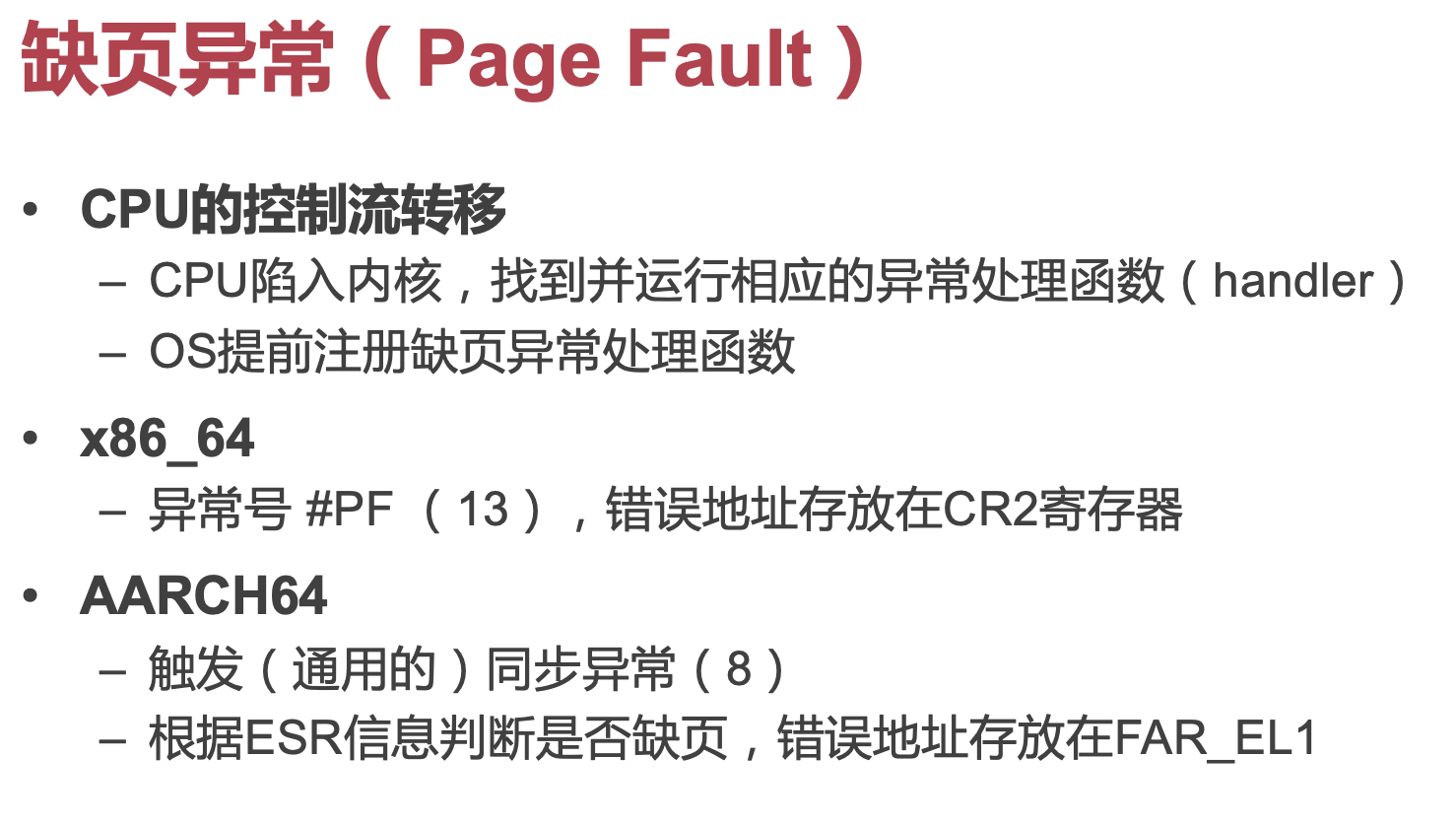
虽然我们可以不用VMA，直接用页表建立完成映射。

用VMA可以实现延迟映射。

## 延迟映射/按需调页

> 举一个延迟映射，按需调页的例子？

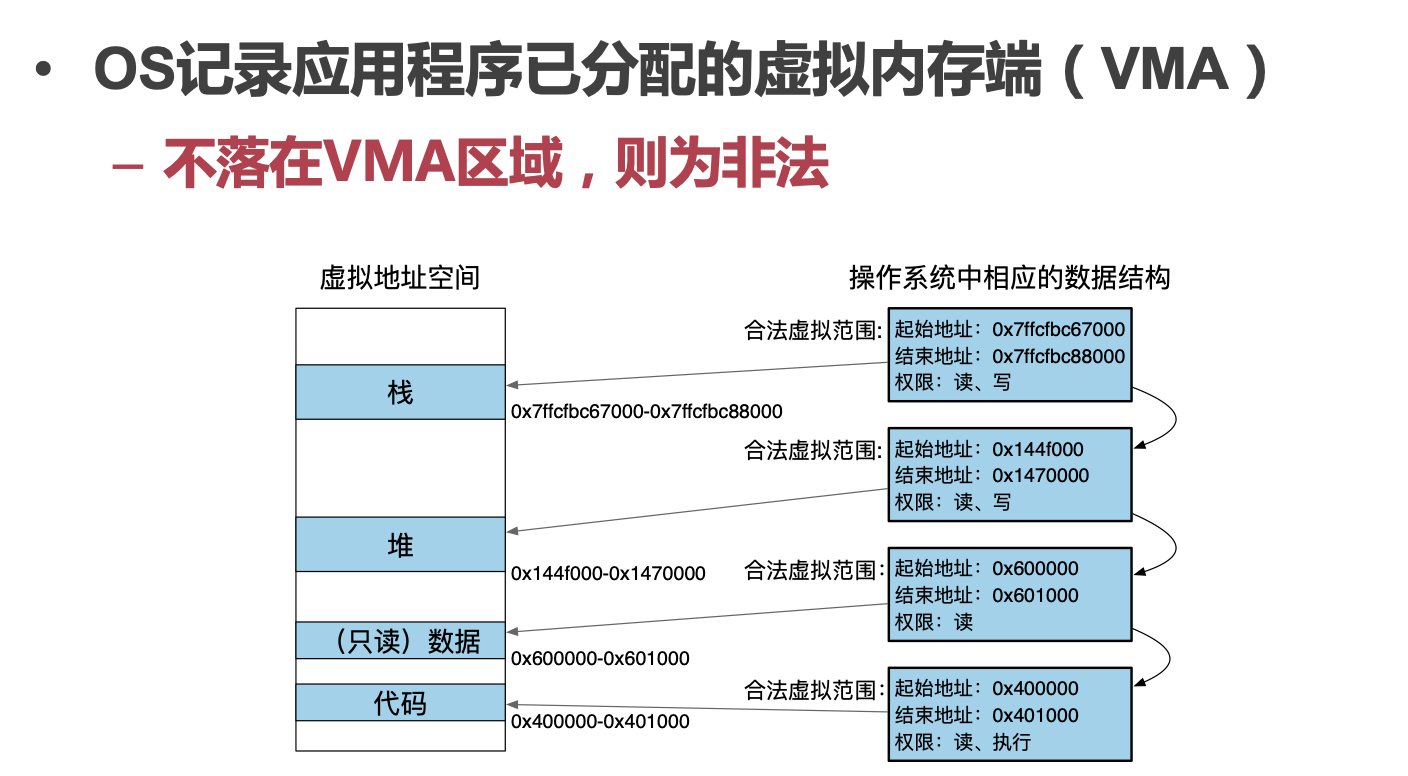
简化加载（From ICS ch9）要把目标文件中的text和data节加载到一个新创建的进程中，Linux加载器为代码和数据段分配虚拟页，把他们标记为无效的（未被缓存的），将页表条目指向目标文件中适当的位置。这之中没有发生任何磁盘到内存的数据复制。每个页初次被引用时，要么是CPU取指，要么是指令引用一个内存位置。



> exception table怎么填的？cpu根据里面对应的handler，直接取调对应的handler。

一般都有约定，比如x86\_64。AArch64，同步异常，编号是8。错误地址放在fault\_address\_el1

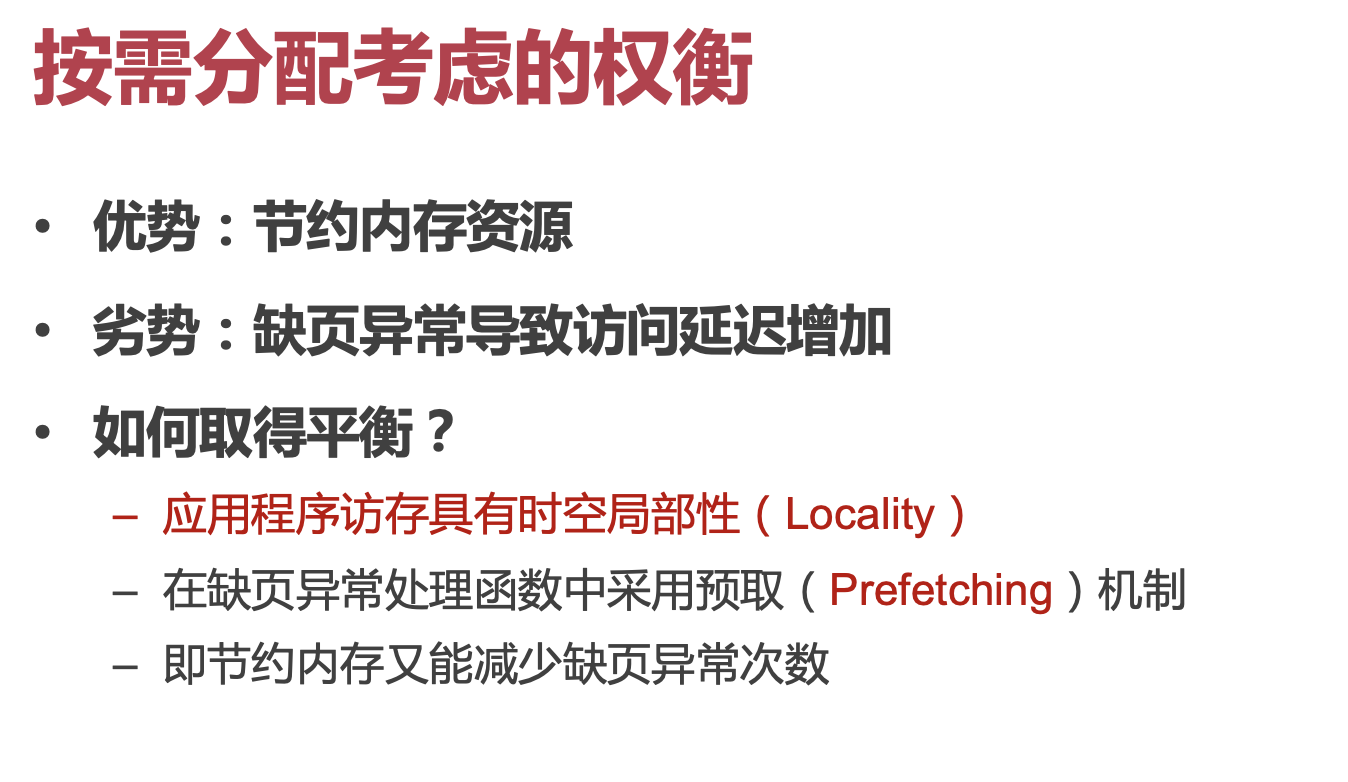
Q：如何判断缺页异常合法性？



如何让访问空指针合法：在0虚拟地址里放一个VMA映射。

cons：对程序运行造成困扰，不可预测行为。

VMA怎么存？少，链表就行，多了，可能要用树，红黑树等。



soft page fault：文件的页在page cache里了

hard page fault：文件对应的数据还在磁盘上，要加载到page cache里面，再建立页表

> 手机第一次打开应用，稍微会慢一点。一直在loading：处理hard page fault

就会出现：把数据从硬盘里面读到内存里建立映射

计算机中广泛应用的概念：locality（局部性）可以prefetch——处理hard page fault的时候：读一个页的数据，根据访问历史，如果访问第一个页，大概率访问第二个页，第三个页，一口气读8个页上来——对磁盘来说，读4k和32k差别没有特别大——之后访问避免page fault

大家可以看一下linux，里面基本都有prefetch机制——prefetch多少个页？根据场景调优，配置——手机和server有很大不同——即节省内存，又减少缺页异常错误。

> maybe use ML to do prefetch

> 都是大家可以设计的空间



用VMA来区分。第一种，不在VMA区域；第二种和第三种，在VMA区域

区分后两种的思路是：

1、记住有哪些页被换出，比如hash表可以解决。

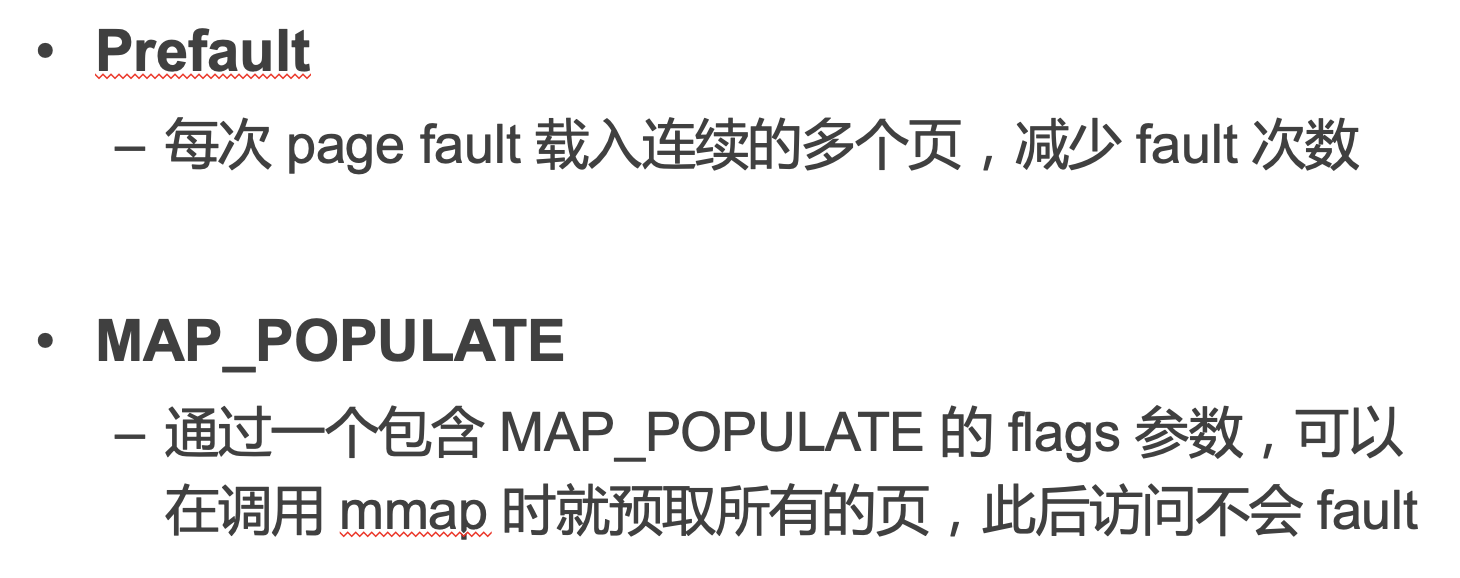
2、更简单的方法：通过页表项某些位是否为NULL进行区分。（OS自己定义规则：NULL代表非法，非NULL代表swap）

> 为什么要区分后两种？后两种不是一样吗？

（From TA Yuchao Qian）按需分配有两种：1. 文件映射，这种情况确实跟第三种差不多，内核里可以复用操作。2. 匿名映射，此时是没有硬盘上的空间的，也没有物理页，page fault时会再物理页中分配一个新的，并且将其初始化为0。

操作系统接口提供了一些机制，接口上的标记，让应用告诉操作系统，如何管理内存？

MMAP优化



mmap完了一块区域，map完之后，应用马上把所有页访问一遍，不如告诉操作系统，你帮我把所有页都取进来。

理解了OS的机制，可以更好的使用OS。

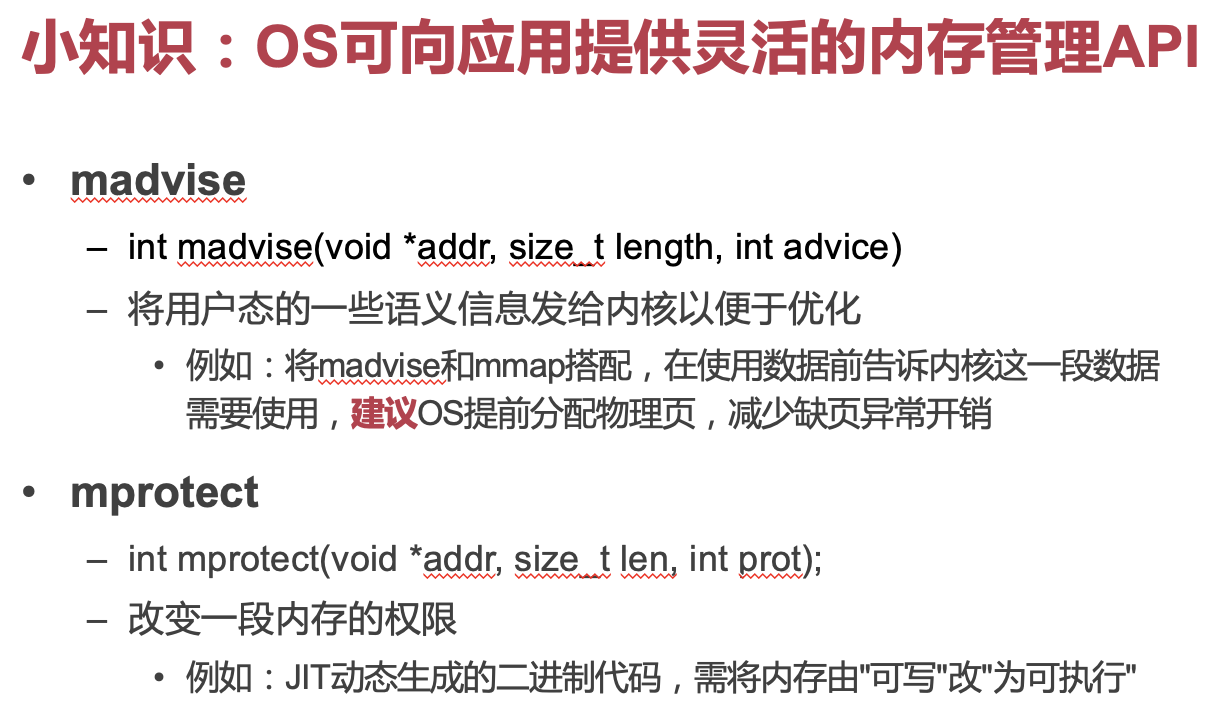
> 13' location base service，根据你的位置推送信息。地理信息系统比赛（GIS cup），算法上类似，经度类似，因为mmap用得好，速度快。



fd标记成-1没有对应的file desctipor，没有后备文件。问-1：初始值为0

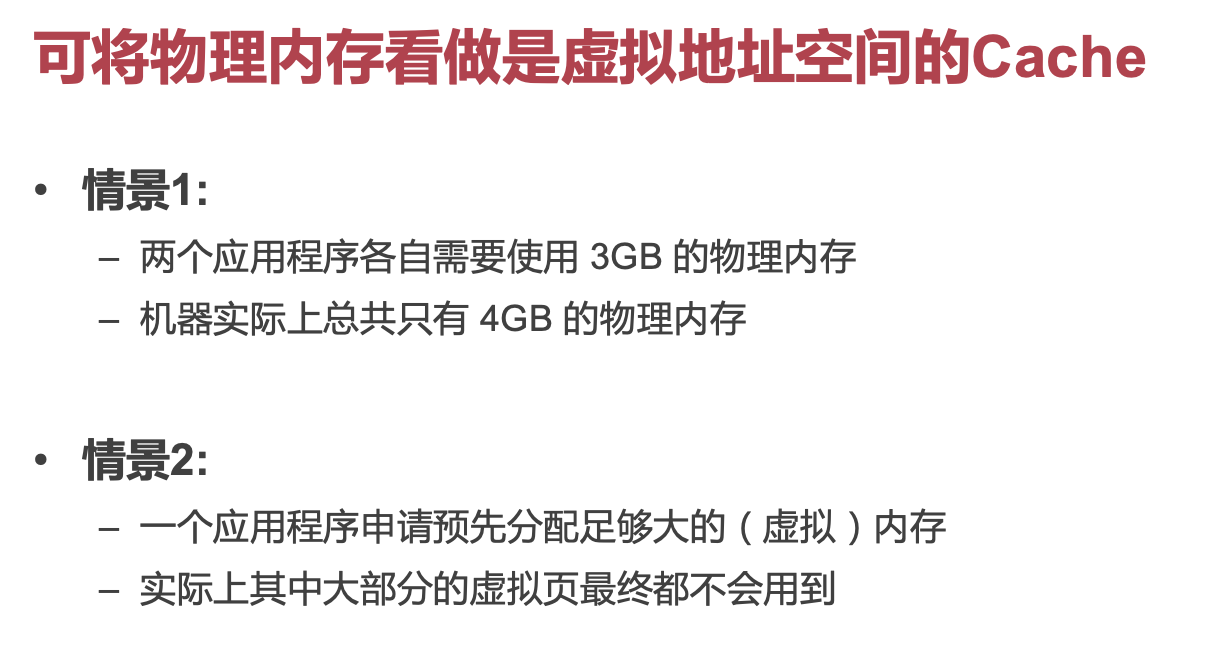
问-2：可以不需要，但性能会变差，VMA查找更方便 VMA可以记录的信息更多：匿名映射？文件back等等。

问-3：可以通过网络，不需要磁盘，实现无盘工作站（网络启动，a protocal called PXE）

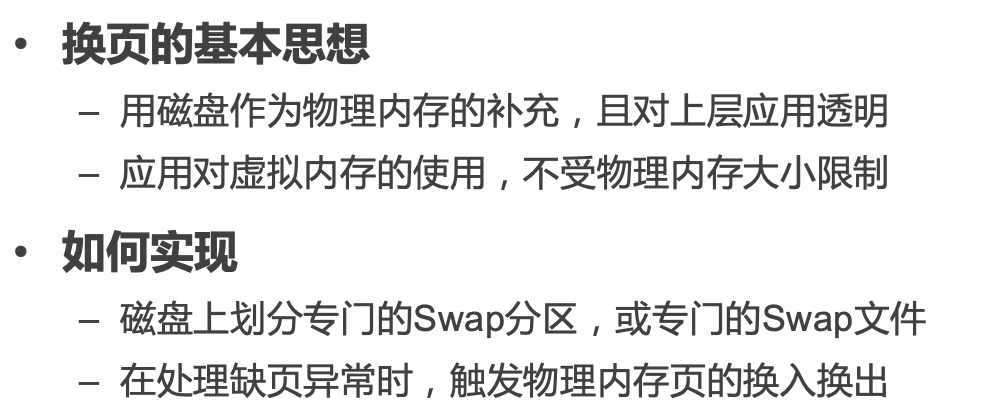


1. 告诉OS内存访问模式：随机，连续等

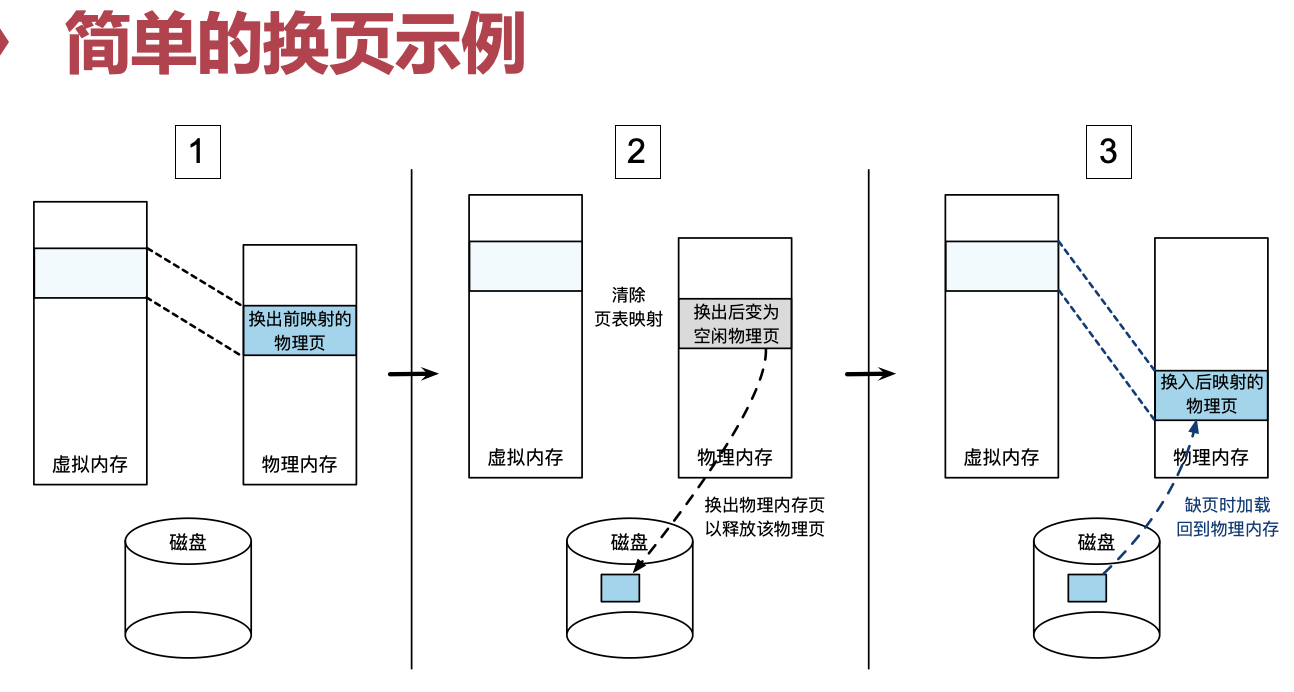
2. just in time翻译成二进制代码



## 换页机制



linux，windows等都通过swap文件

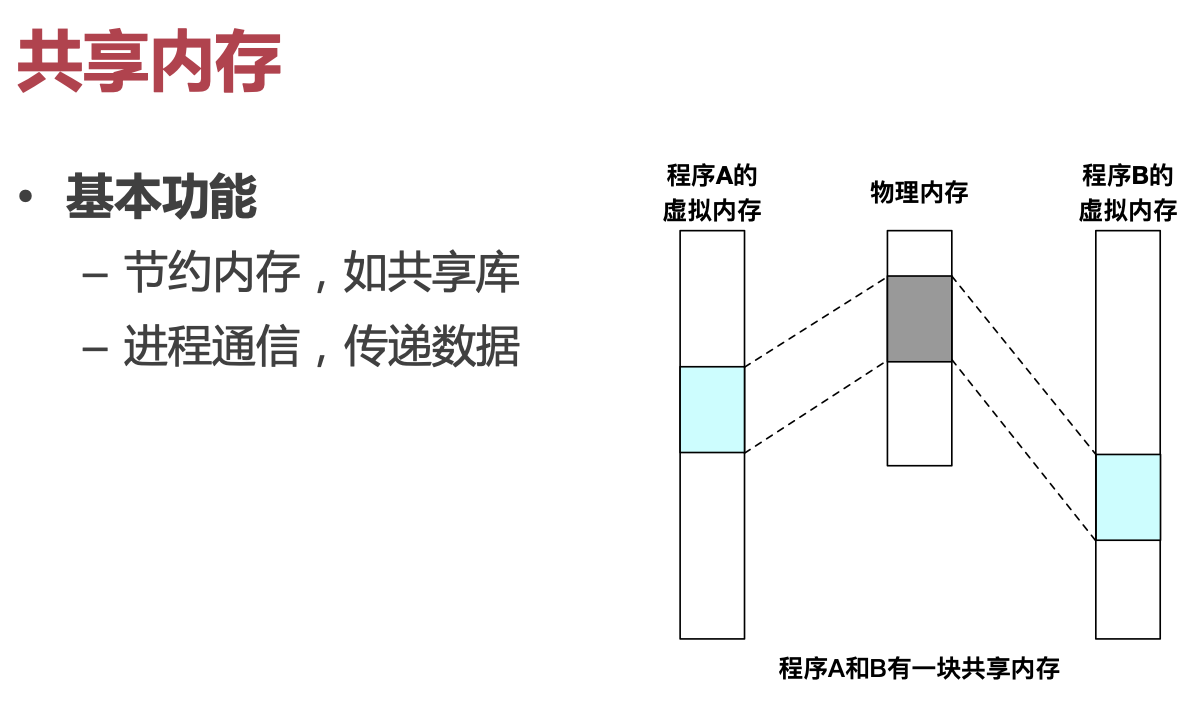


Q：手机里内存不够怎么办？

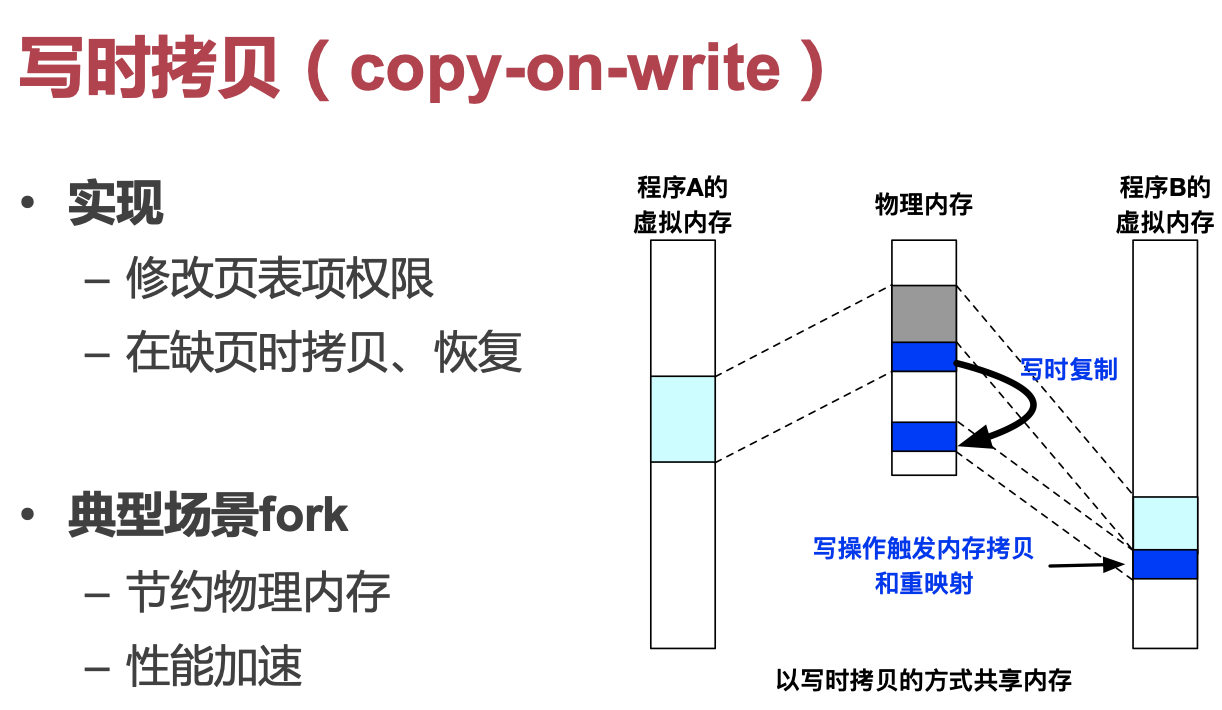
A：杀后台。杀后台的后果：微信打开防疫健康码，手机验证，切回去之后不见了。

也可以采用换出的机制。（Huawei前年发布的，可以让你8G的内存等价于12G

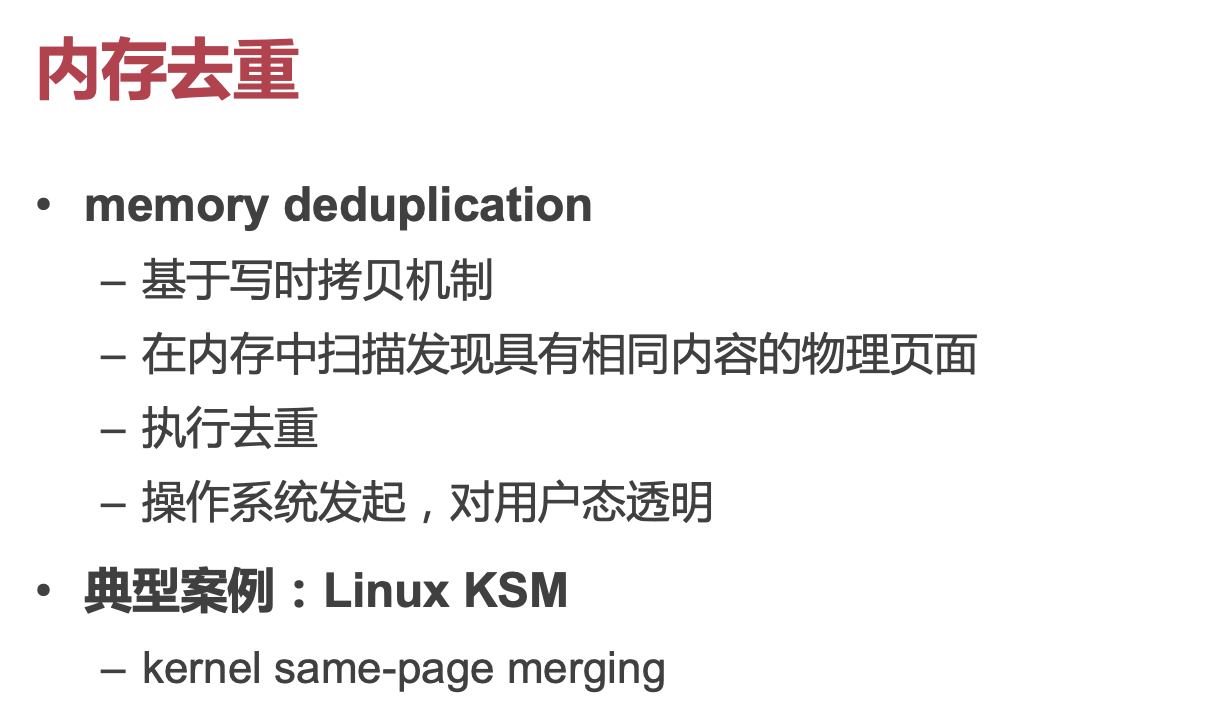
## OS内存管理更多机制



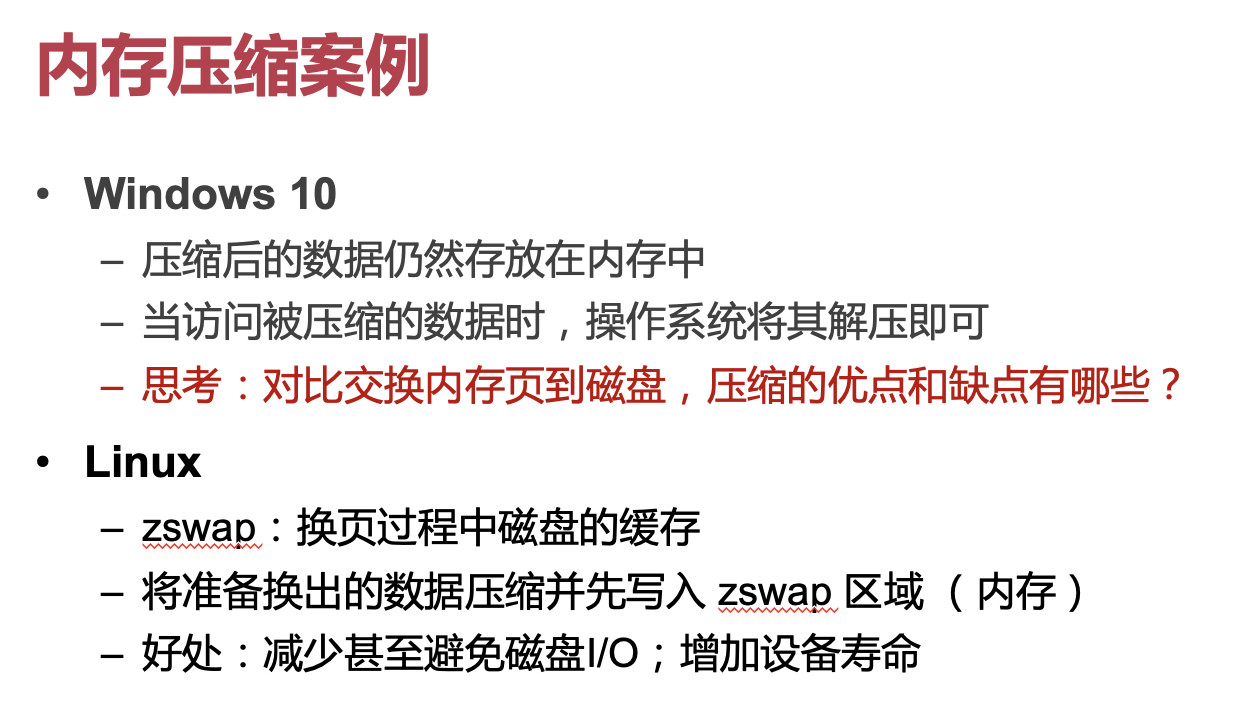
典型：fork



> 批判fork：代价很大：要创建页表



手机里打开文件几次，用不同的应用打开。其在云中：很多虚拟机大量页是重复的。如果有大量物理页就COW。扫描一遍，去重。KSM副作用：导致侧信道攻击。



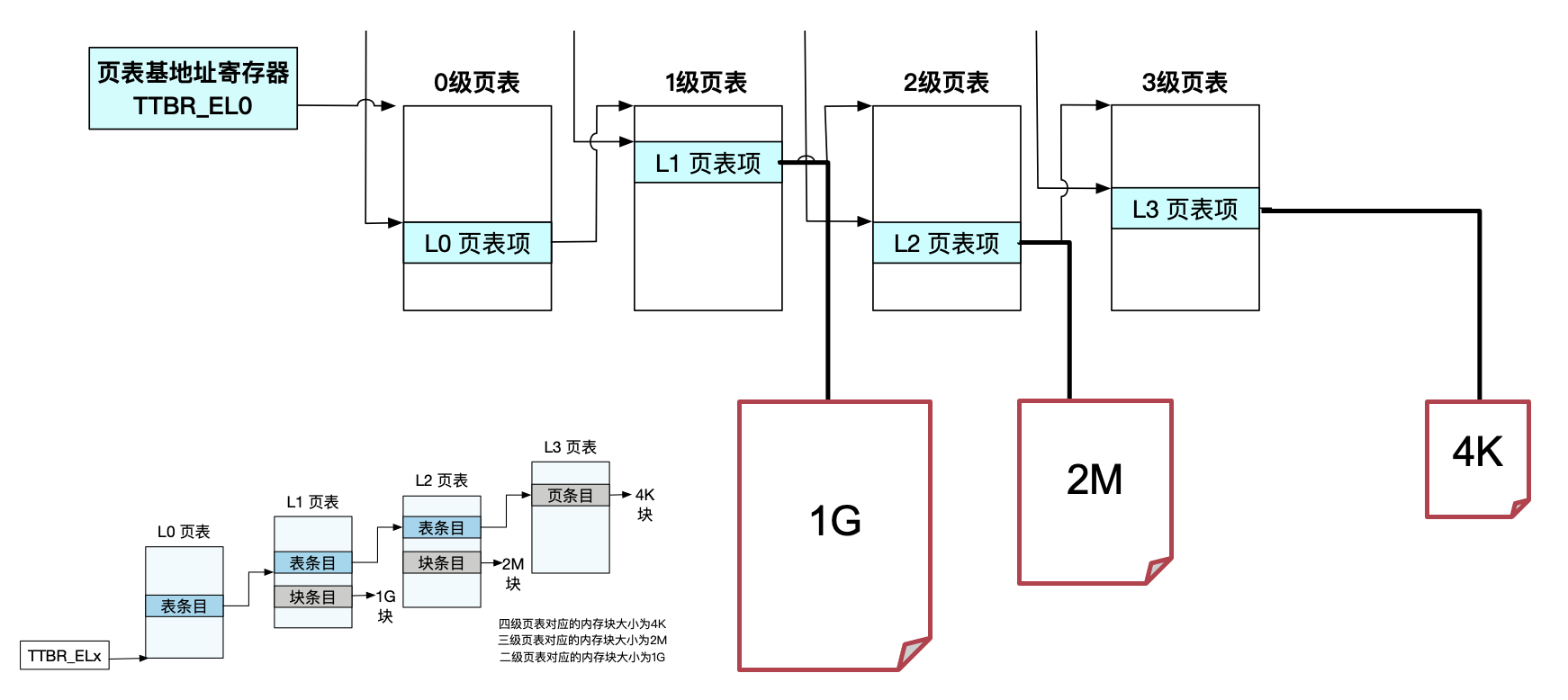
缺点：增加CPU负担。

优点：IO省掉了。

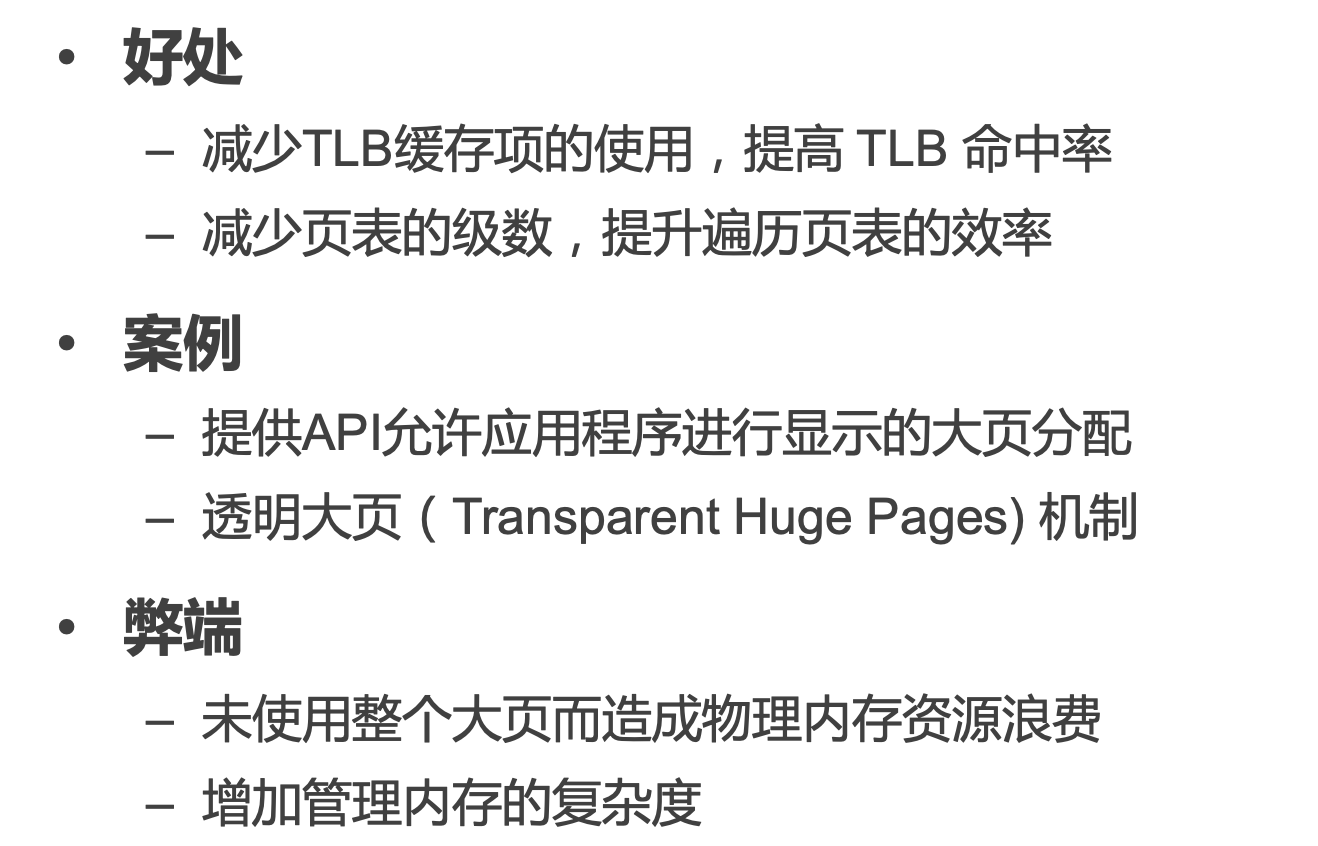


huge page：4级页表，有些页表项值保留两级或三级页表





理论上L0：512G也可以。



要用1G大页，内存里必须要有1G连续的空间。

THP：存在了很多年，但没有真正使用——核心原因：增加管理内存复杂性。



Q：为什么只有L2支持大页？

A：

苹果上：16k

Q：什么情况适合使用大页？

A：访问相对连续，内存充足。大数据的场景，更少的TLB miss

Q：在物理内存足够大的今天，虚拟内存是否还有存在的必要？

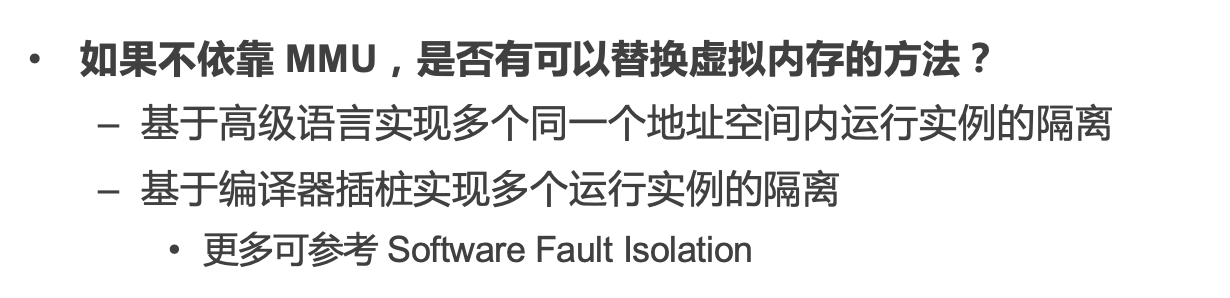
- 如果不使用虚拟内存抽象，恢复到只用物理内存寻址，会带来哪些改变？哪些场景适合？

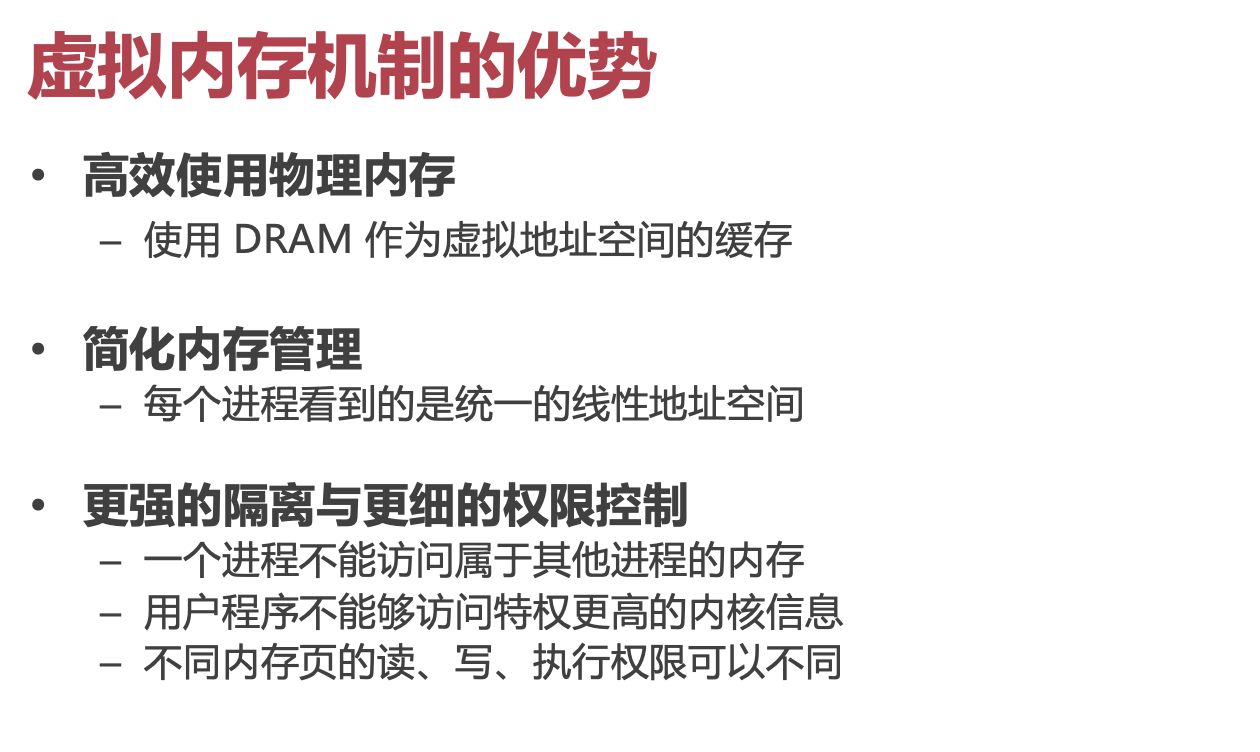
1. 车载系统，MCU。

2. 如果只用物理内存，不适合多用户，适合单用户。

VM除了按需换页，还有保护，还有抽象，使编译解耦，使得写的程序可以跨平台运行。

当今研究：虚拟内存是否一定要页表：TB级内存：页表占用太大，大数据场景，仍然用segment机制等等。





企业：大的系统定位定界。VM建立了很好的隔离机制。如果都在内核里，定位定界会变得很难。

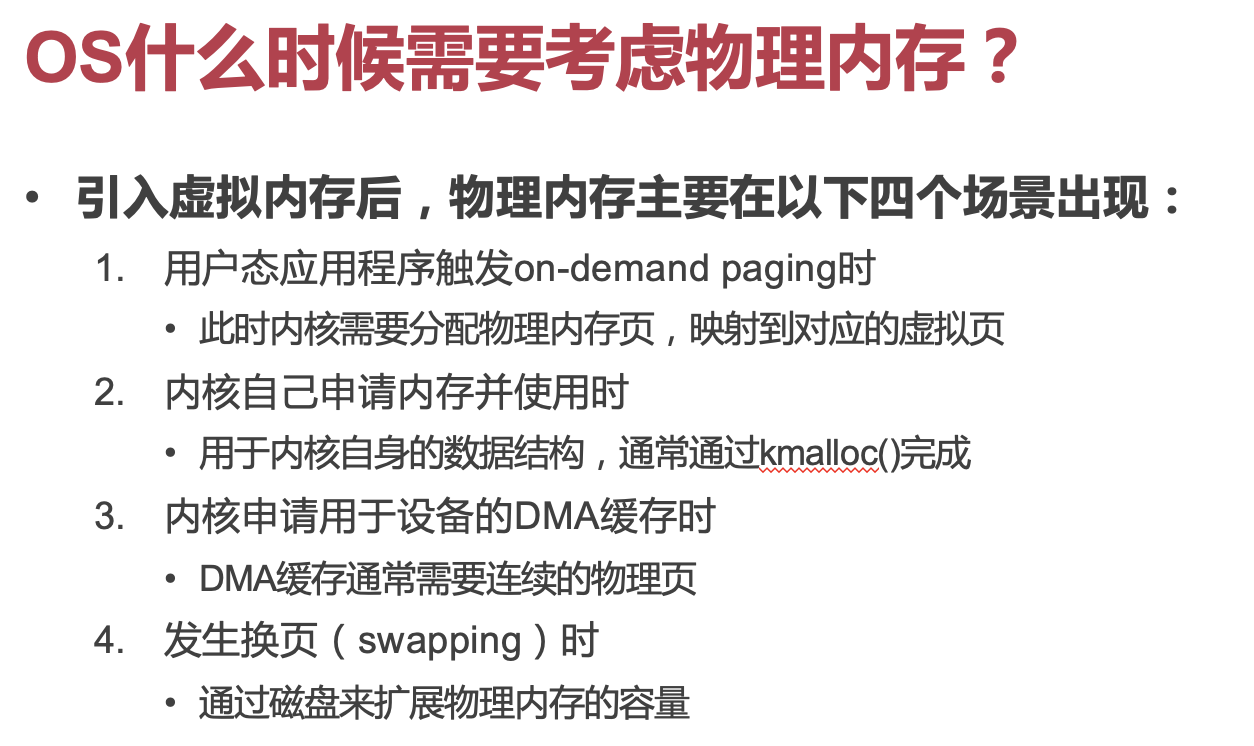
## 物理内存管理

Q：物理地址切换到虚拟地址保护做了哪些事情？

虽然是物理地址管理，OS运行时用的还是虚拟地址。管理的还是虚拟地址，只不过虚拟地址和物理地址做了一一映射。

如果出现缺页异常，实际上分配到是虚拟地址，然后会把virtual to physical的转换，然后填入页表。

> 凡是给CPU看的，就是物理地址。

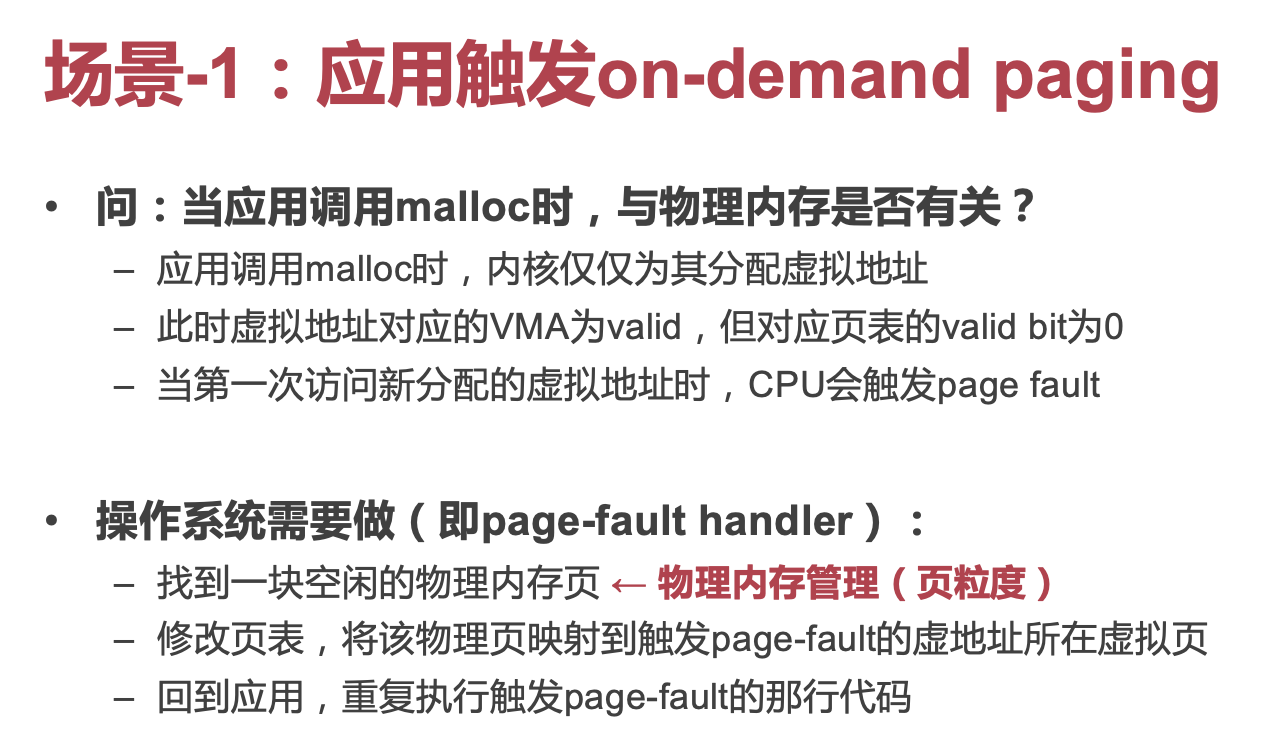


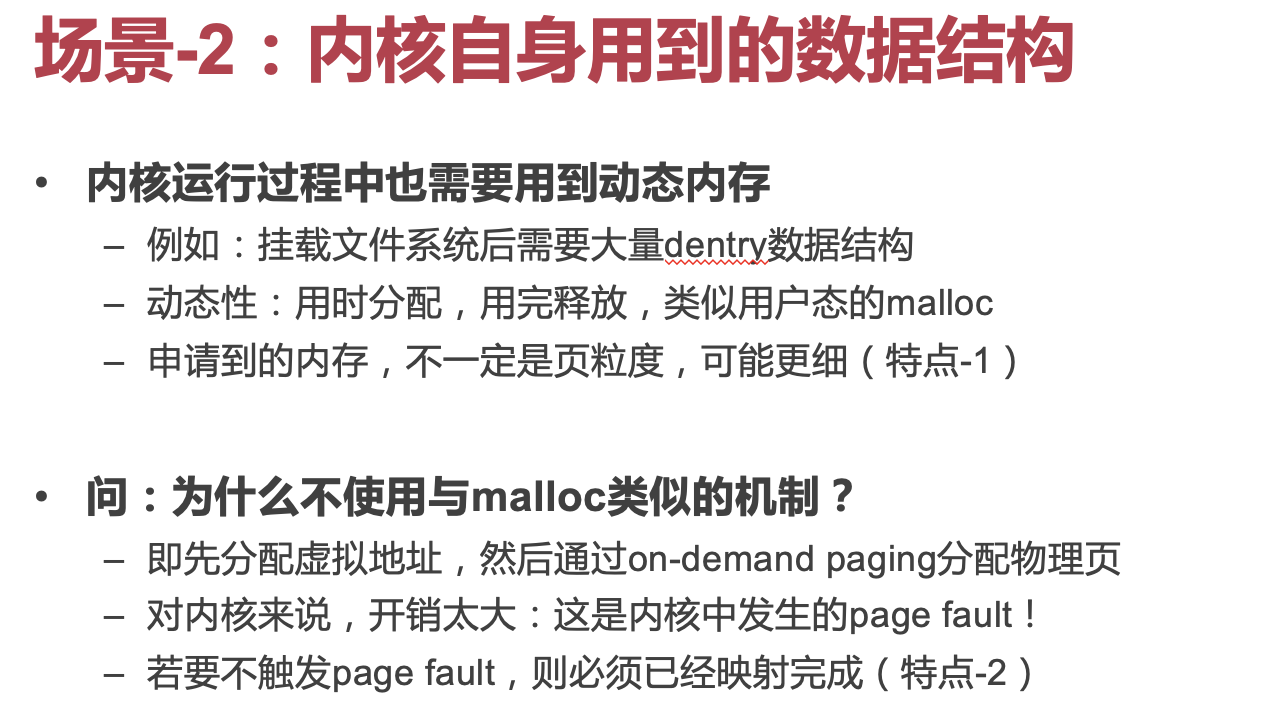
DMA：direct memory access

现在设备都支持：scatter gather，可以不连续，设备用的时候再汇聚起来。

Q：开一个相机：需要多少内存？

A：拍照的话2G起步。

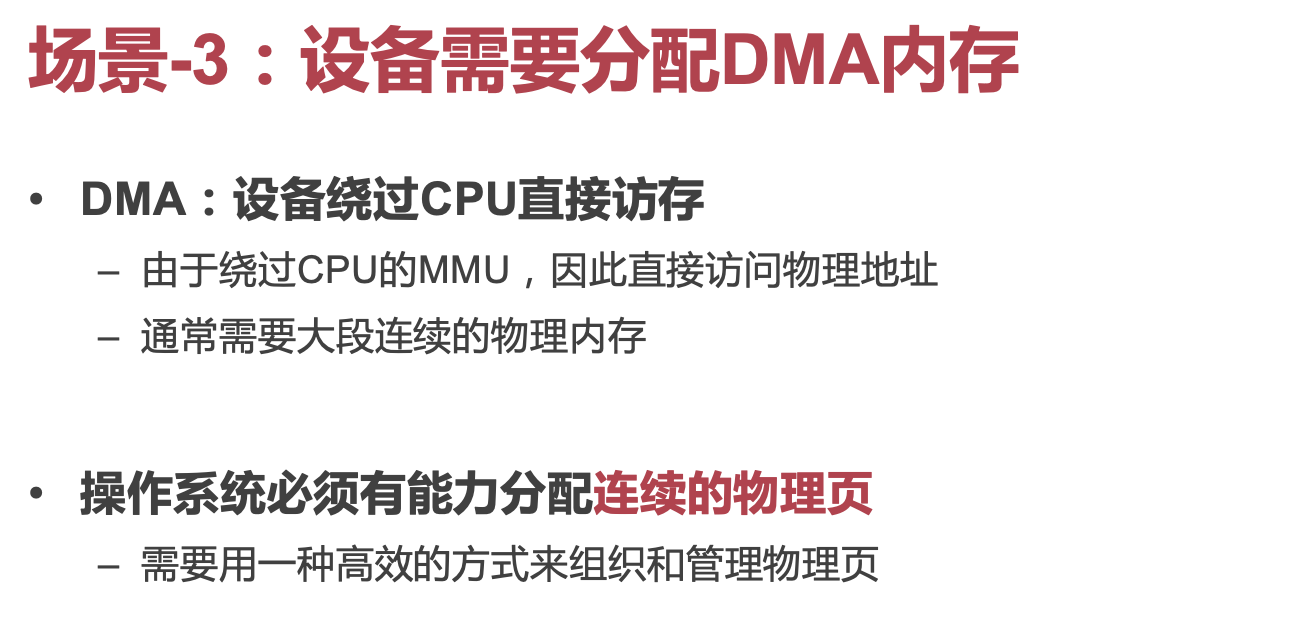


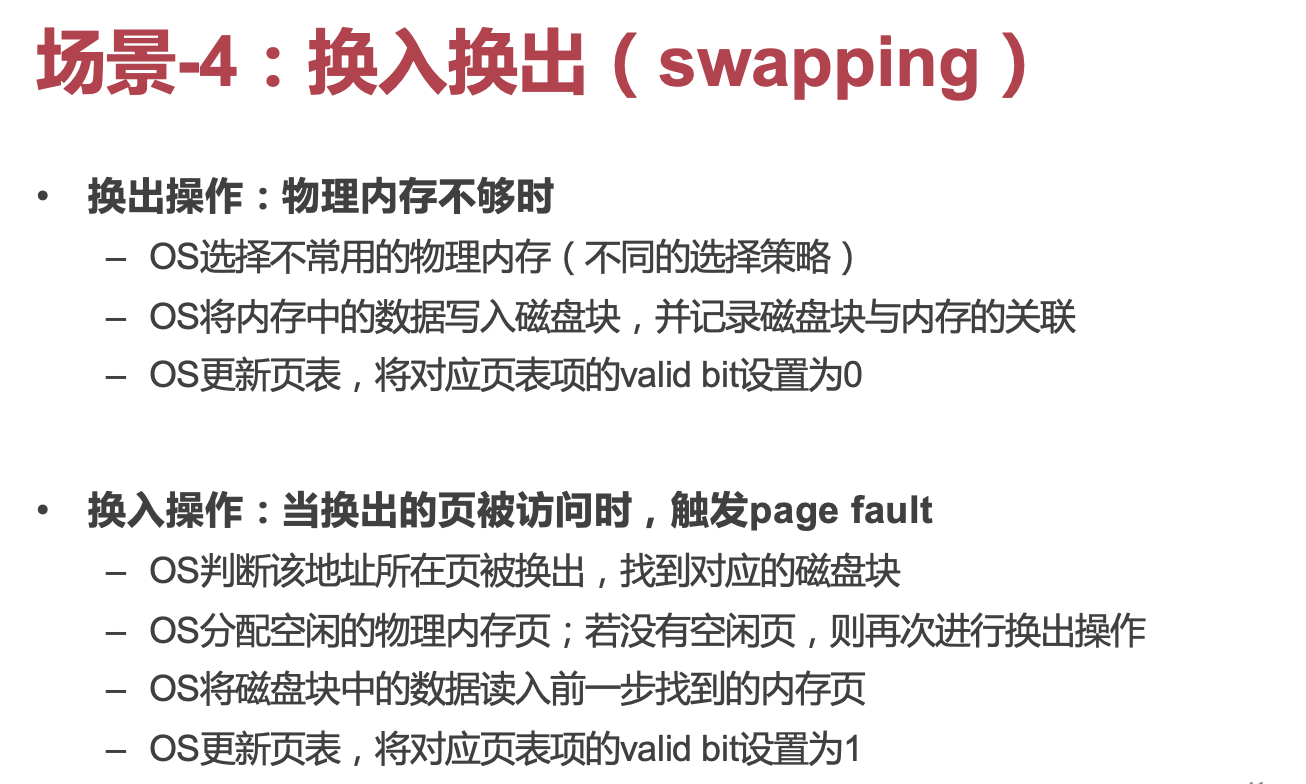


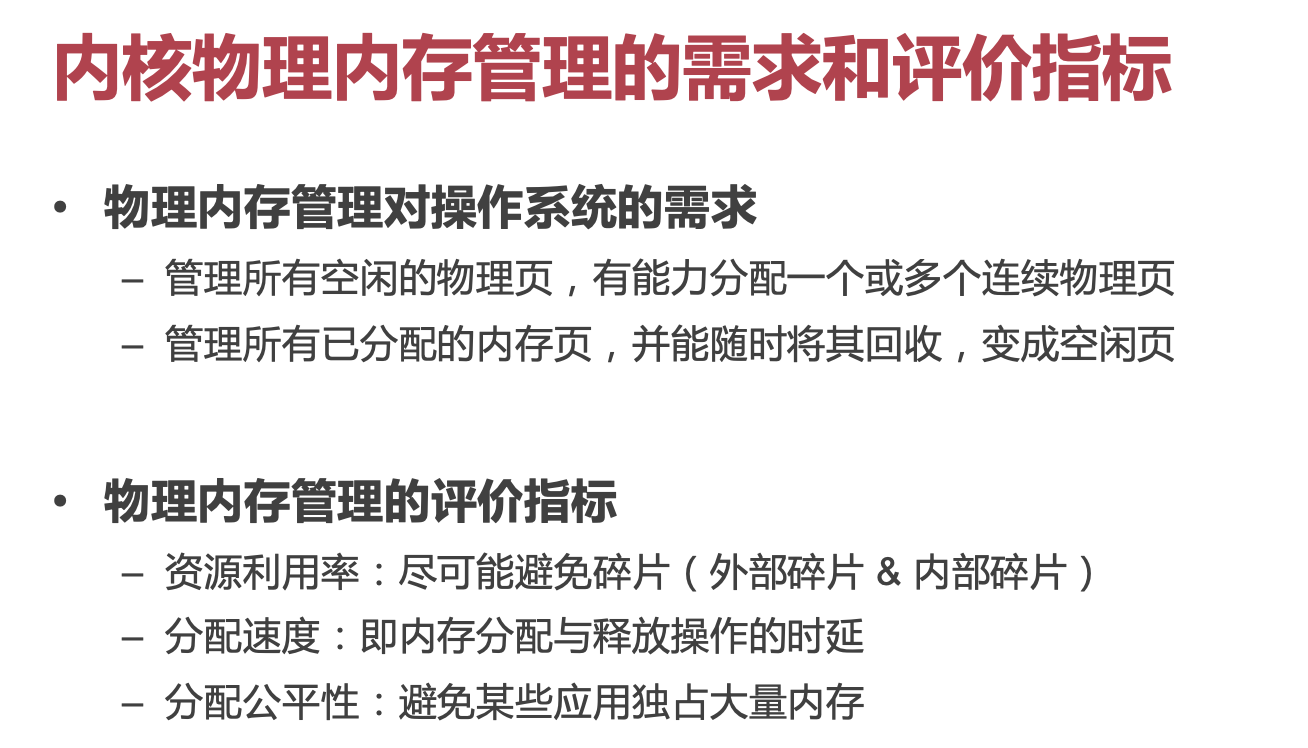
dentry：directory entry

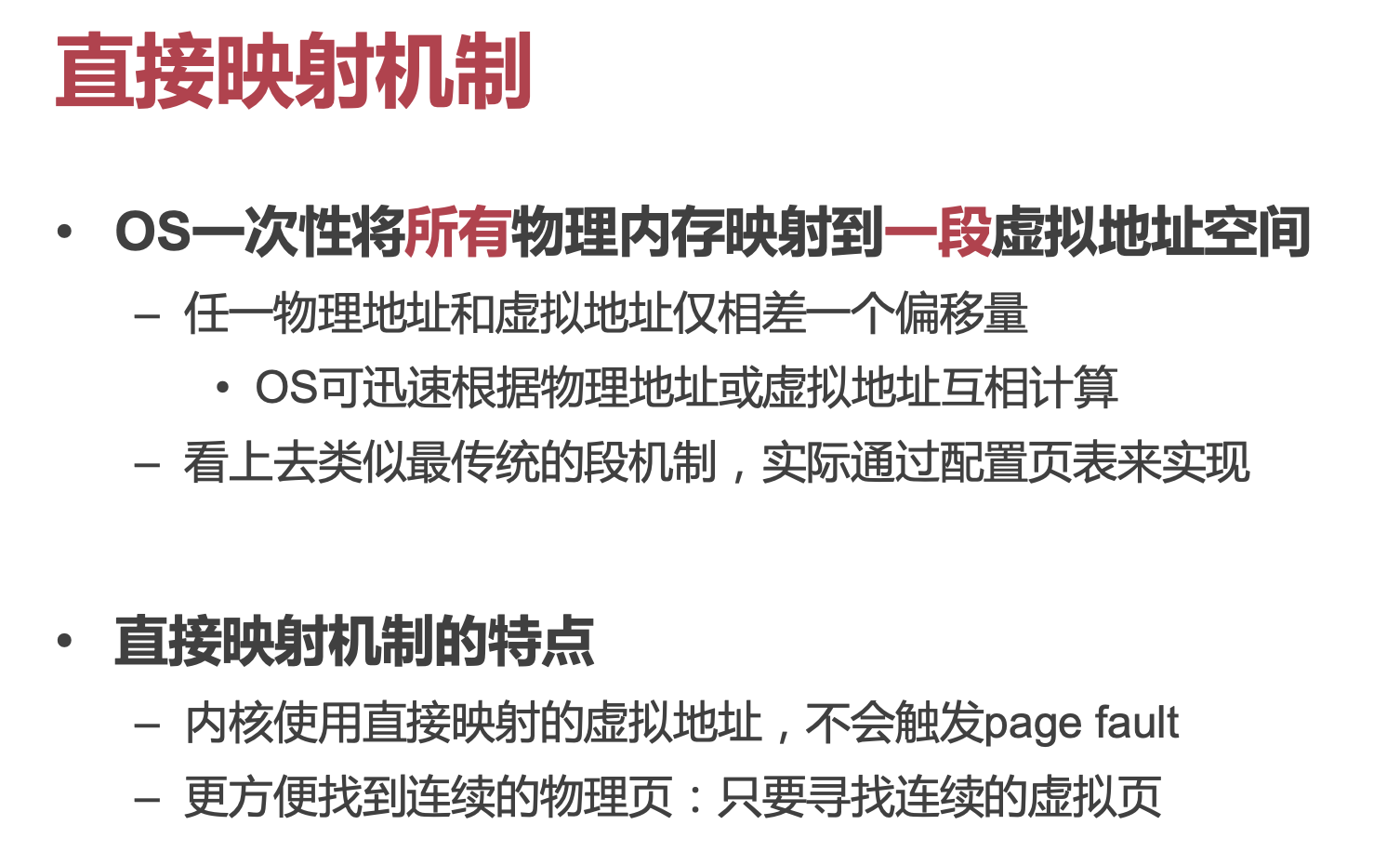
dentry大概几百个bytes，用不了4k

> OS两个流派：新泽西（bell实验室，worse is better），MIT（尽善尽美）









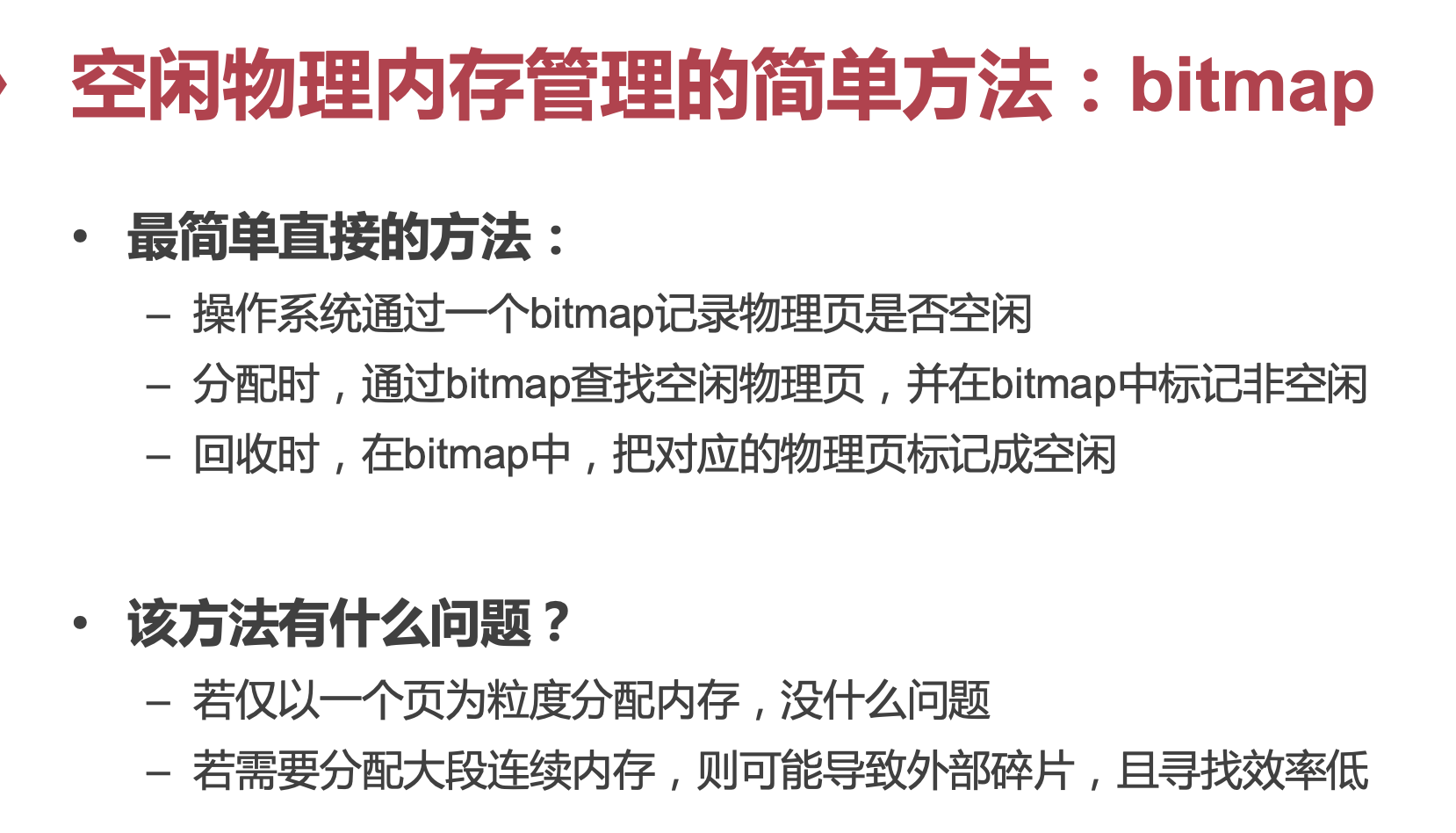
内核虽然是在管物理地址，但是用虚拟地址操作的。



vmalloc：先分配一块虚拟内存，再填上物理内存。——造成kernel中有page fault。

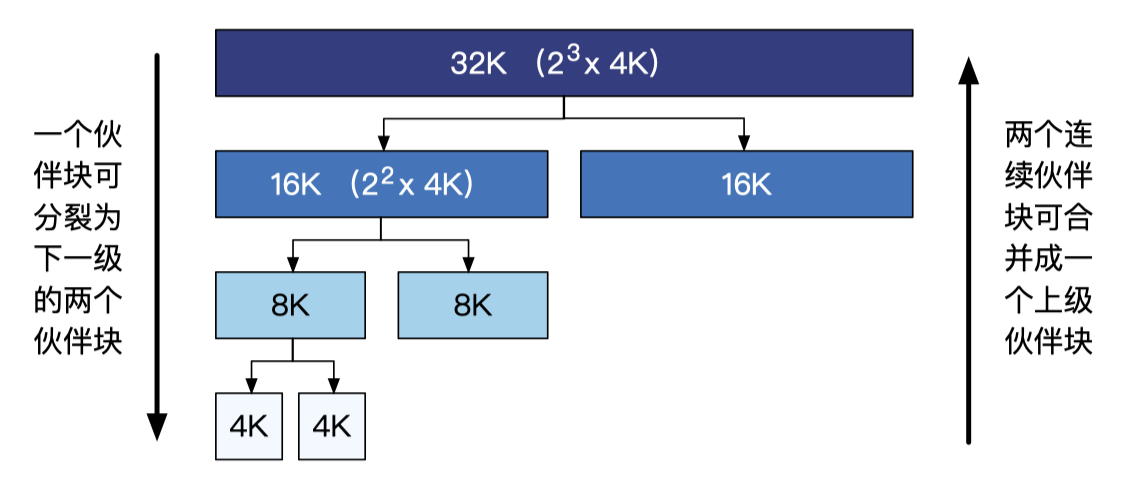
启动的时候就将物理地址至少应用了两遍。

## 伙伴系统：页粒度内存管理

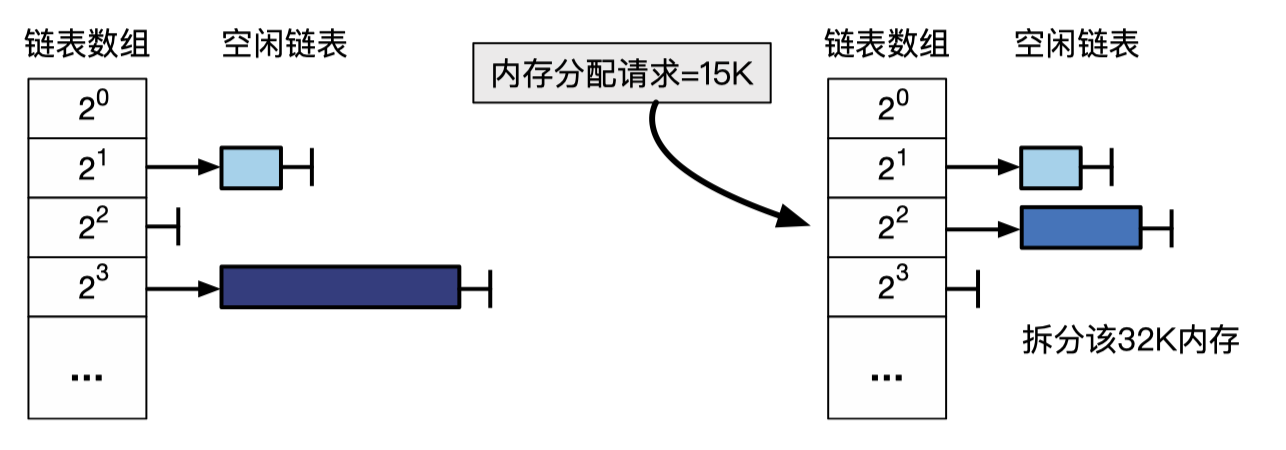


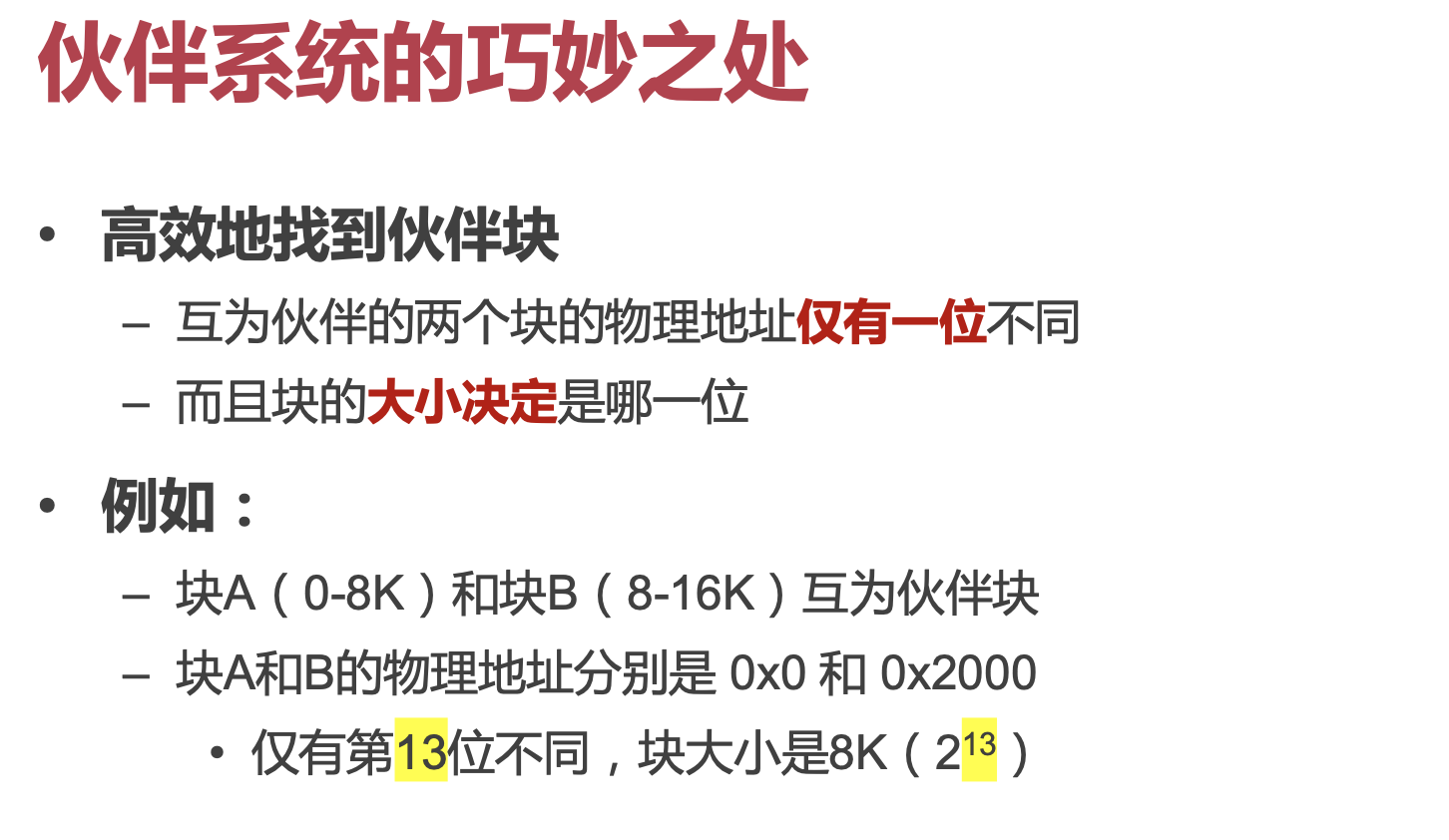
cons：相机，视频播放，需要大块空间，并不是仅仅以4k为粒度进行分配。导致外部碎片

### buddy system

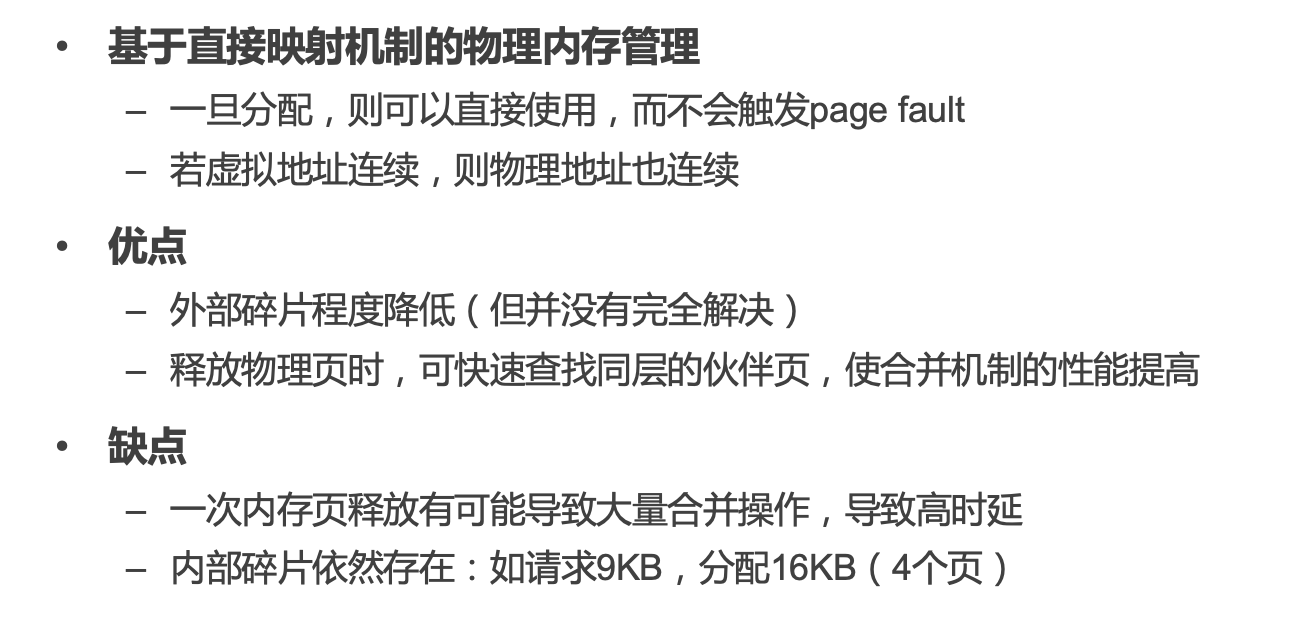


连续两个块就可以合并，对于kernel来说，分裂和合并非常高效。



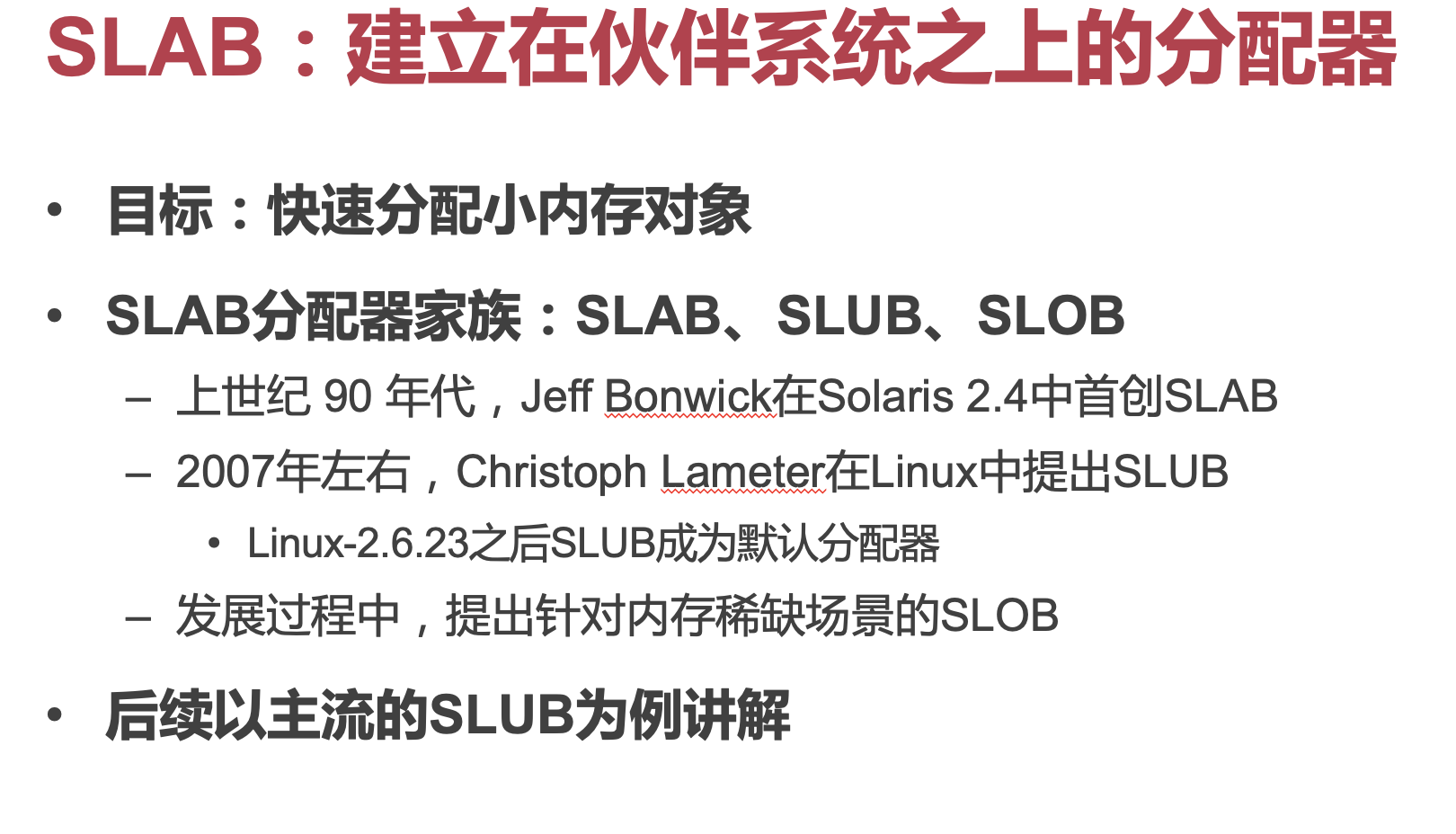


buddy sys pros & cons



### 如何实现内核的linux？

对于OS而言，写好了内核，需要哪些数据结构已经确定了，可以为常用数据结构定制特定的“块”。



slab：链表

slub：没有用链表，用的是更高效的机制

slob：内存稀缺型

