

# 操作系统 *Operating System*

## 第三章 内存管理-页目录自映射

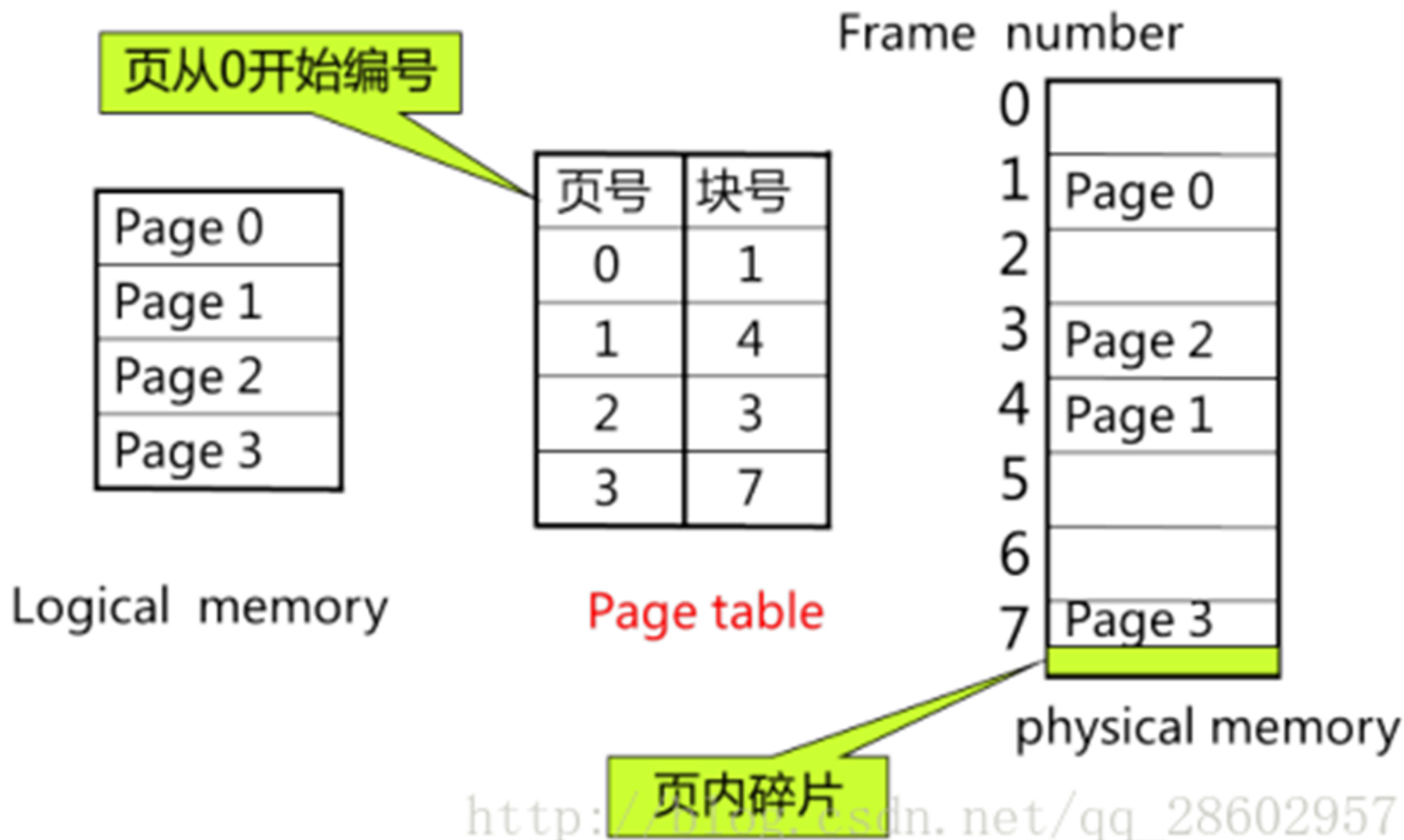
沃天宇

woty@buaa.edu.cn

2024年3月29日

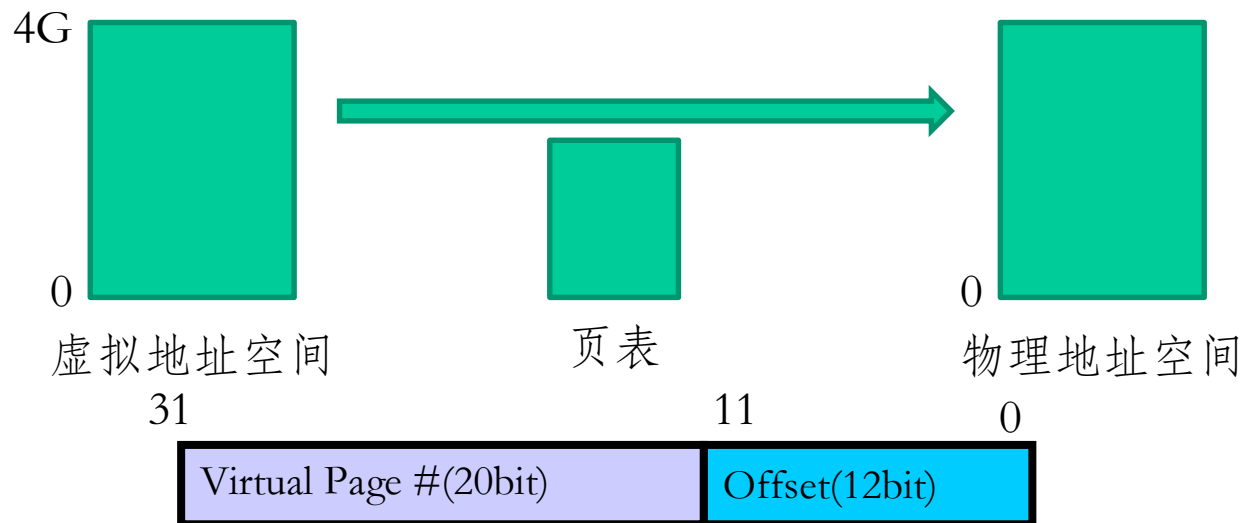
# 回顾：页式内存管理

- 页式内存管理思想：破除内存分配“连续性假设”



# 回顾：页式内存管理

- 页表的作用是将虚拟地址空间映射到物理地址空间



- 对于32位地址长度，可寻址空间为4GB
- 采用12位页内偏移，表明内存页大小为4KB
- 每个页表项负责记录1页（4KB）的地址映射关系
- 整个4GB地址空间被划分为 $4GB/4KB=1M$ 页，所以需要1M个页表项来记录逻辑-物理映射关系

# 一级页表的问题

## ■ 页表的大小与逻辑地址空间大小成正比

- 如果逻辑地址空间很大，则划分的页比较多，页表就很大，占用的物理存储空间大，实现较困难（分配困难）。
- 例如：对于 32 位逻辑地址空间的分页系统，如果规定页面大小为 4 KB 即  $2^{12}$  B，则在每个进程页表就由高达  $2^{20}$  页组成。设每个页表项占用 4 个字节，每个进程仅仅页表就要占用 4 MB 的连续物理内存空间。
- 64位逻辑地址空间呢？

–  $2^{64}/2^{12}=2^{52}$  页， $2^{54}$ B=16PB的连续物理内存

4G



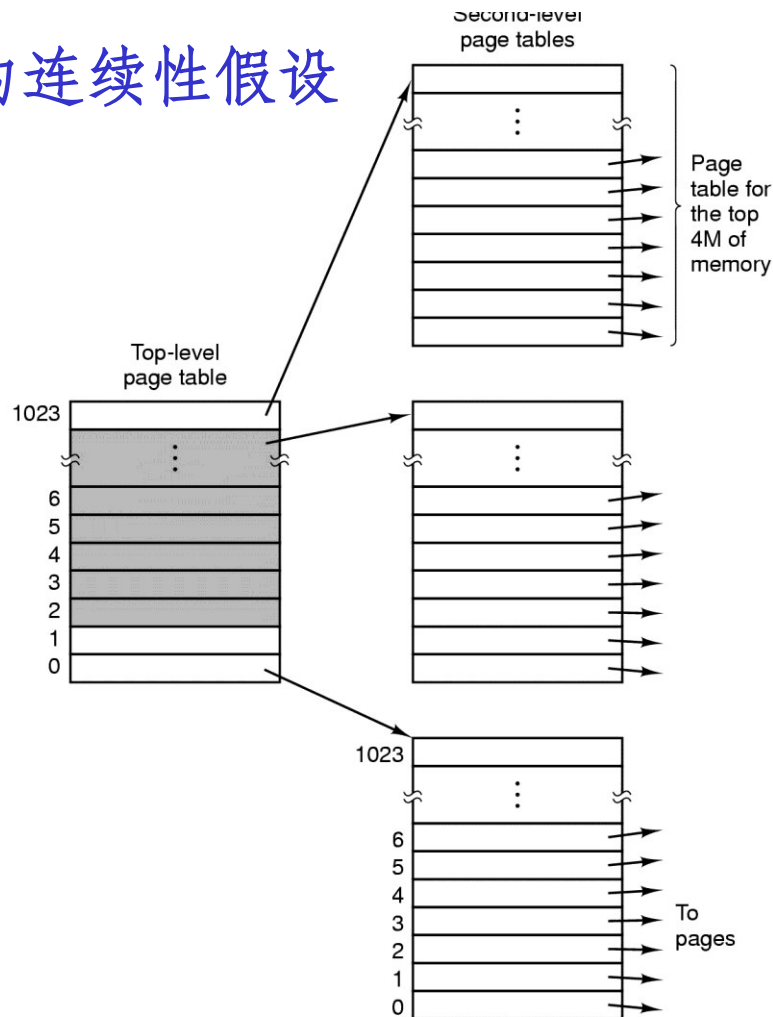
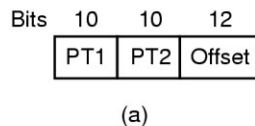
# 多级页表

## ■ 解决问题的方法

- 多级页表：继续破除页表存储的连续性假设
- 动态调入页表：只将当前需用的部分页表项调入内存，其余的需用时再调入。

## ■ 例：

- 32位地址空间
- 4K页面大小
- 1M+1K页表项
- 地址结构  
— 10 10 12



# 基本功练习1

- $1 \text{ KB} = \underline{1024} \text{ B} = 2^{10} \text{ B}$

- $4 \text{ KB} = \underline{4096} \text{ B} = 2^{12} \text{ B}$

- $1 \text{ MB} = 2^{20} \text{ B}$

- $4 \text{ MB} = 2^{22} \text{ B}$

以下用最大字节单位

- $2^{16} \text{ B} = \underline{64 \text{ KB}}$

- $2^{32} \text{ B} = \underline{4 \text{ GB}}$

- $2^{64} \text{ B} = \underline{16 \text{ EB}}$

- $2^{10} \text{ B} = 1 \text{ KB}$

- $2^{20} \text{ B} = 1 \text{ MB}$

- $2^{30} \text{ B} = 1 \text{ GB}$

- $2^{40} \text{ B} = 1 \text{ TB}$

- $2^{50} \text{ B} = 1 \text{ PB}$

- $2^{60} \text{ B} = 1 \text{ EB}$

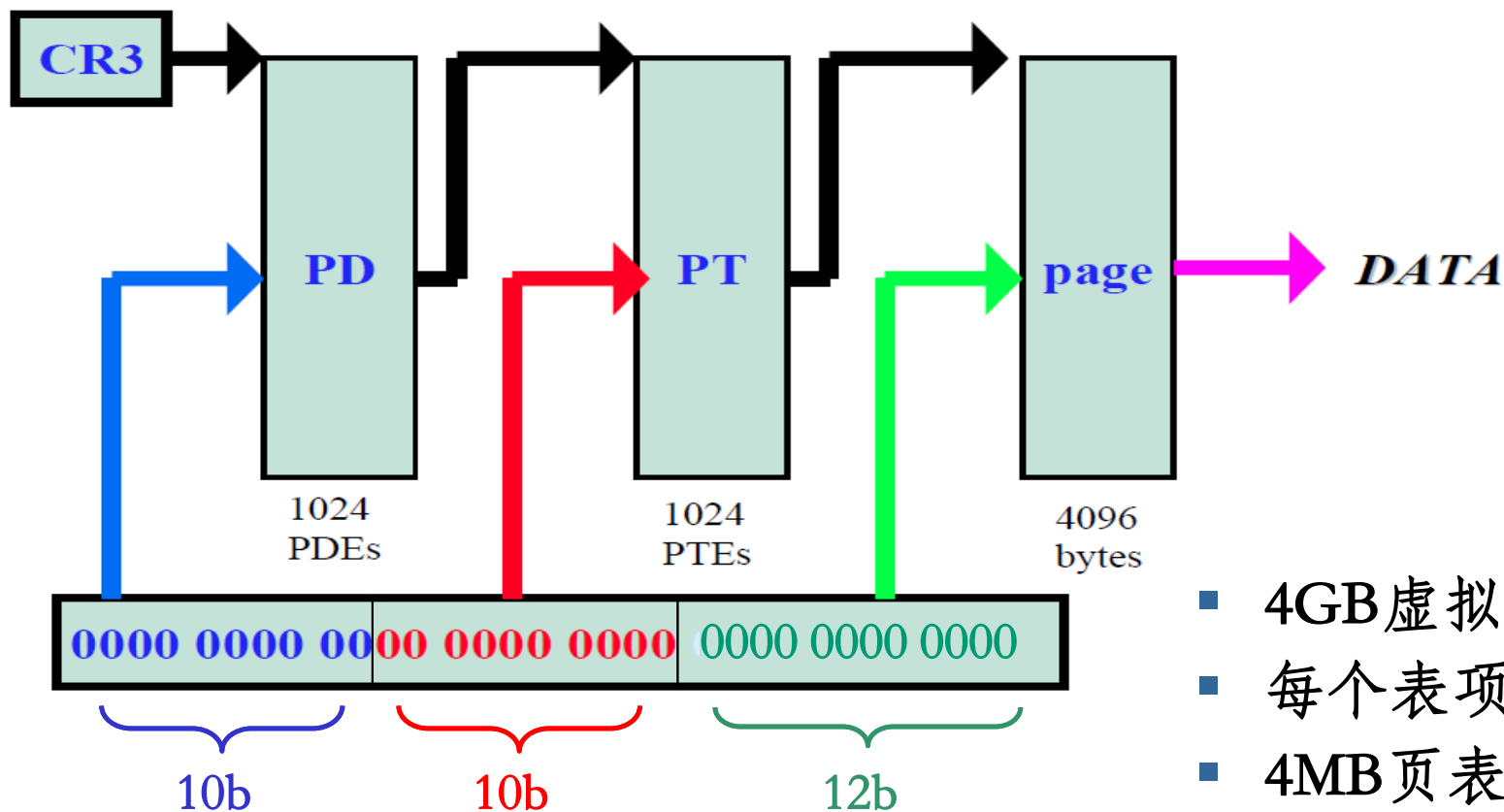
- $2^{70} \text{ B} = 1 \text{ ZB}$

# 基本功练习2

- 十六进制数0x12345678转成二进制有 32 位
- $0x12345678 \gg 12 = \underline{0x12345}$
- $0x80000000 \gg 22 = \underline{0x200}$
- $0x1000 = \underline{4096}_{(10)} = \underline{4} \text{ K}$
- $16\text{K} = 0x \underline{4000}$  ;  $64\text{K} = 0x \underline{10000}$
- $1\text{M} = 0x \underline{100000}$  ;  $4\text{M} = 0x \underline{400000}$
- $1\text{G} = 0x \underline{40000000}$  ;  $2\text{G} = 0x \underline{80000000}$
- $3\text{G} = 0x \underline{C0000000}$  ;  $4\text{G} = 0x \underline{100000000}$

# 多级页表

## Virtual Address Translation

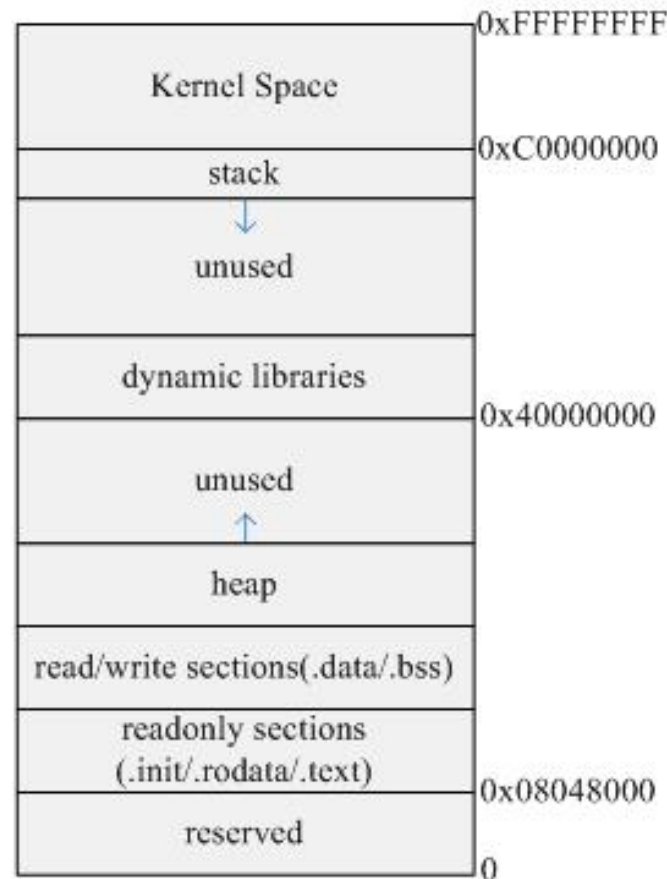


- 4GB虚拟地址空间
- 每个表项4个字节
- 4MB页表项(PTEs)
- 4KB页目录表项(PDEs)



# 页表管理

- 谁来管理（填写）页表？
  - 当然是OS
- 填写页表目的？
  - 反映内存布局
- 如何填写、修改页表？
  - 写页表所在内存
  - 用虚拟地址还是物理地址？

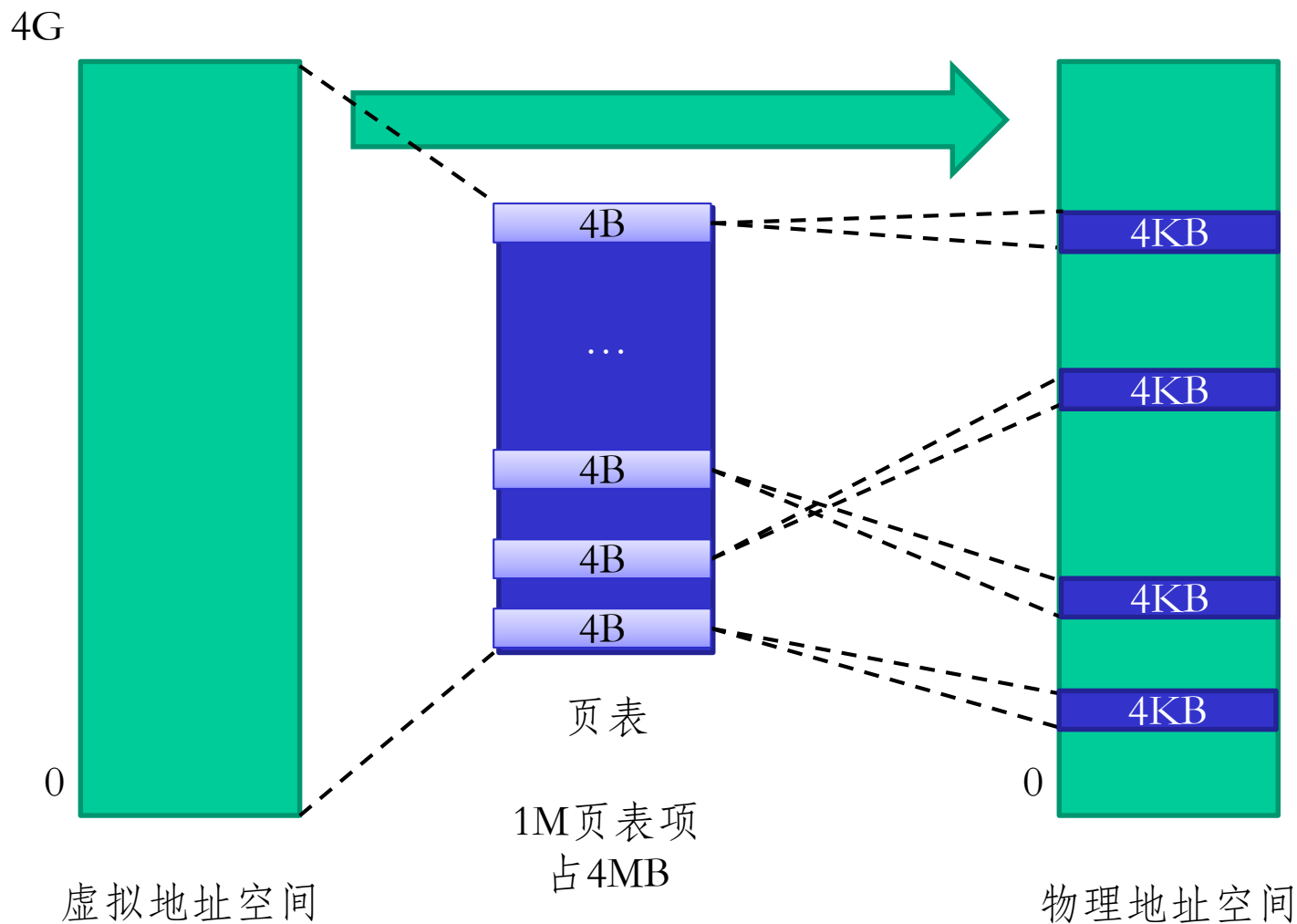


```

9      o      /      Invalid memory      /      /\
10     o      +-----+-----+-----+-----+-----Physics Memory Max
11     o      /      ...      /      kseg0
12     o      VPT,KSTACKTOP-----> +-----+-----+-----0x8040 0000-----end
13     o      /      Kernel Stack      /      / KSTACKSIZE      /\
14     o      +-----+-----+-----+-----+-----/
15     o      /      Kernel Text      /      /      PDMAP
16     o      KERNBASE -----> +-----+-----+-----0x8001 0000 /
17     o      /      Interrupts & Exception      /      \/\
18     o      ULIM -----> +-----+-----+-----0x8000 0000-----
19     o      /      User VPT      /      PDMAP      /\
20     o      UVPT -----> +-----+-----+-----0x7fc0 0000 /
21     o      /      PAGES      /      PDMAP      /
22     o      UPAGES -----> +-----+-----+-----0x7f80 0000 /
23     o      /      ENV$      /      PDMAP      /
24     o      UTOP,UENVS -----> +-----+-----+-----0x7f40 0000 /
25     o      UXSTACKTOP -/      /      user exception stack      /      BY2PG      /
26     o      +-----+-----+-----+-----+-----0x7f3f f000 /
27     o      /      Invalid memory      /      BY2PG      /
28     o      USTACKTOP -----> +-----+-----+-----0x7f3f e000 /
29     o      /      normal user stack      /      BY2PG      /
30     o      +-----+-----+-----+-----+-----0x7f3f d000 /
31     a      /      /      /
32     a      /      /      /
33     a      .      .      .
34     a      .      .      .      kuseg
35     a      .      .      .
36     a      /-----+-----+-----+-----+-----/
37     a      /      /      /
38     o      UTEXT -----> +-----+-----+-----/
39     o      /      /      /      2 * PDMAP      \/\
40     a      0 -----> +-----+-----+-----+-----+-----
41     o
42     */

```

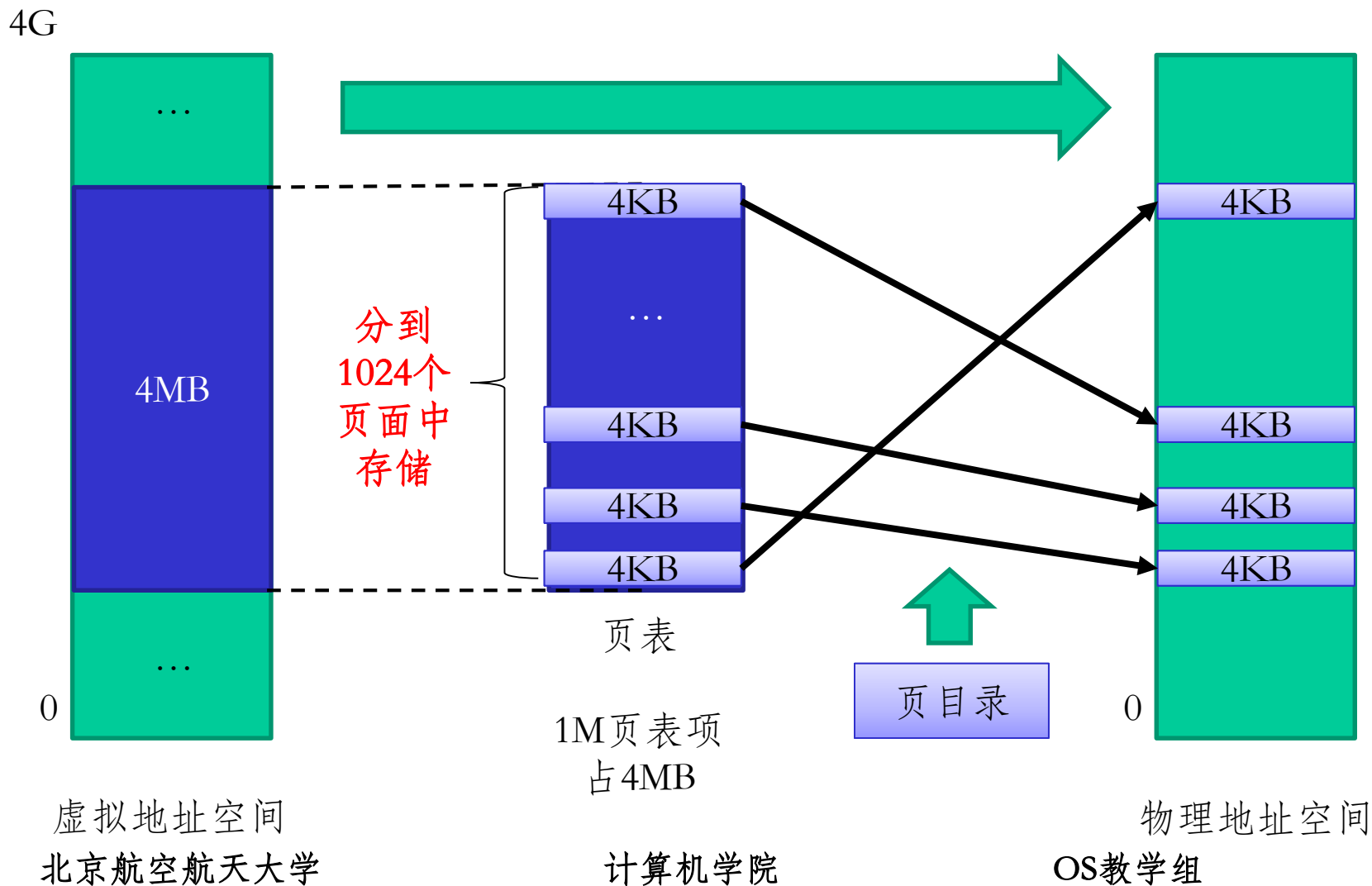
# 页目录自映射



# 页目录自映射

- 页表在虚拟地址空间中映射
  - 每个页表项需要4字节，所以1M个页表项需要4MB，所以整个页表占用的地址空间大小就是4MB
  - 4MB页表也要分页存储，共需要 $4\text{MB}/4\text{KB}=1024$ 个页面存储（页表页）
  - 每一页中存储 $4\text{KB}/4\text{B}=1024$ 项页表项
  - 由于1个页表项对应4KB内存，所以每1个页表页对应 $1024*4\text{KB}=4\text{MB}$ 内存

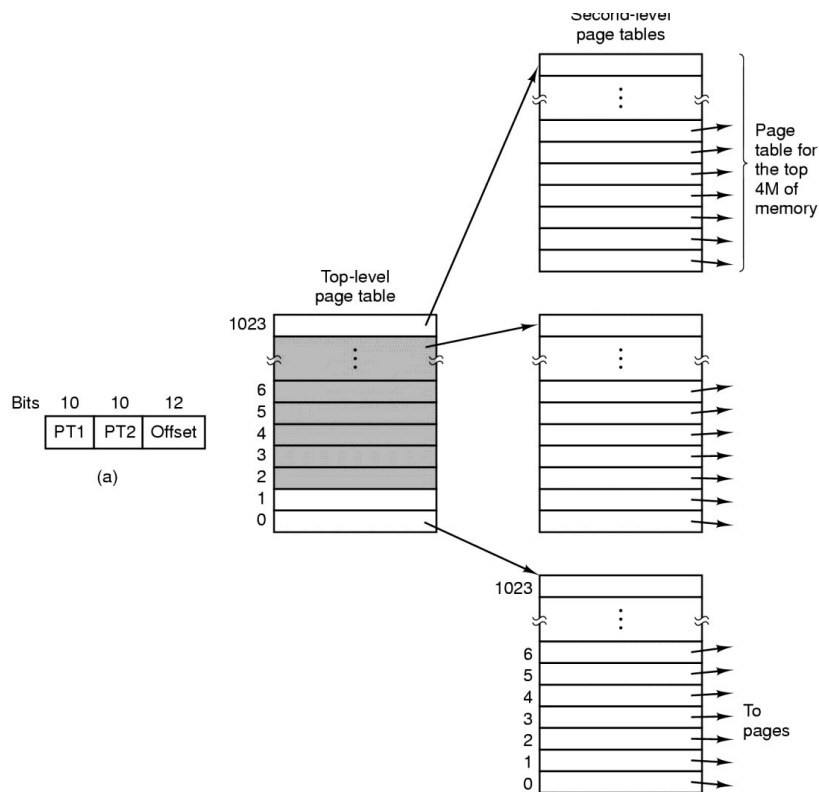
# 页目录自映射



# 页目录自映射

## ■ 页目录定义：页表页的地址映射

- 1024个页表页逻辑上连续，物理上可以分散，其对应逻辑-物理映射关系记录在**页目录**中
- 页目录占1页（4KB）空间，有1024项（页目录项），每一项指向一个页表页
- 每一**页目录项**对应4MB内存，1024个页目录项正好对应4GB内存（整个地址空间）

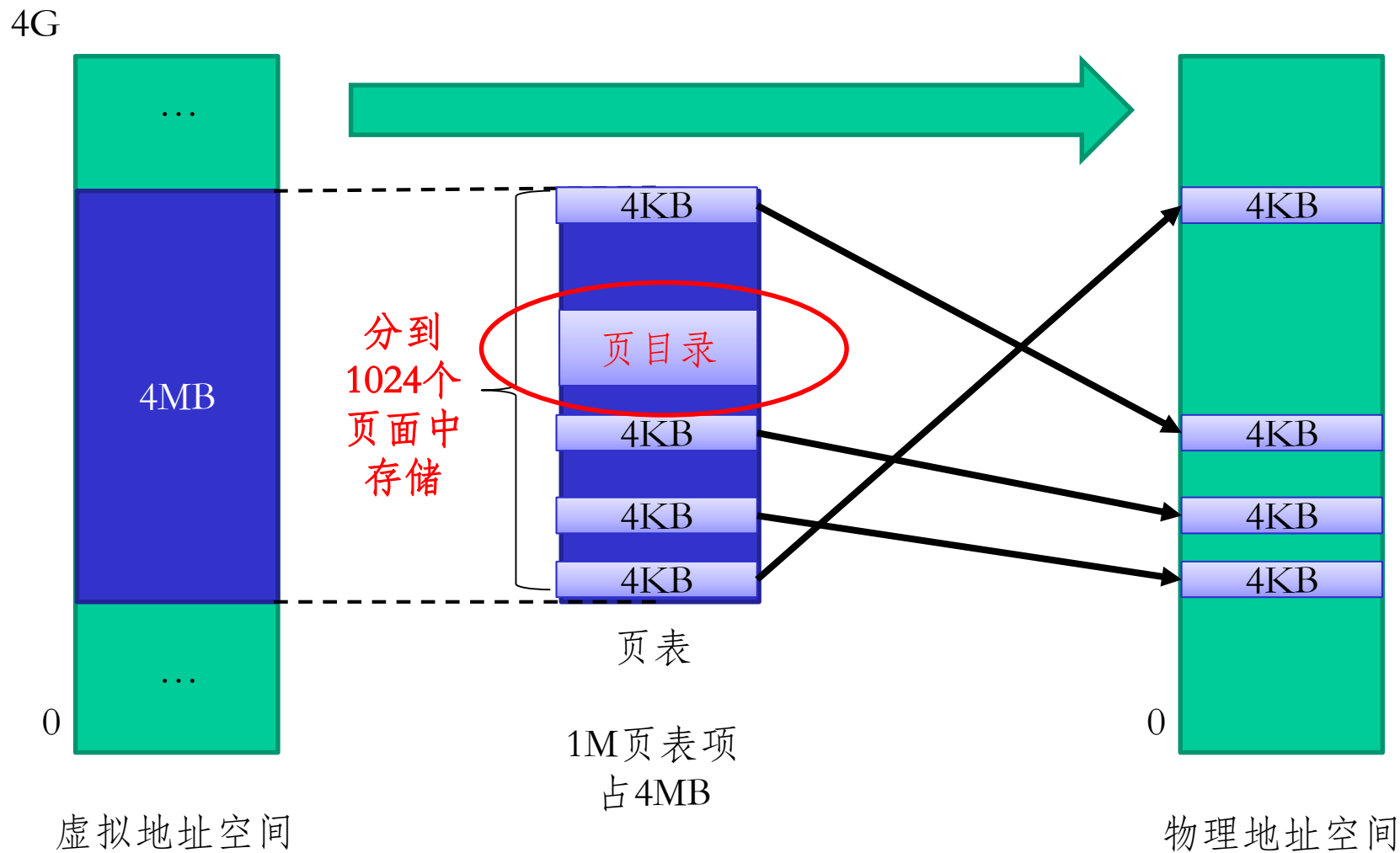


# 页目录自映射

## ■ 关键点

- 存储页表的4MB地址空间中是整个4GB虚拟地址空间中的一部分，OS设计者可规定其所在位置（4MB对齐）
- 一方面根据页目录的定义：记录这4MB（连续）地址空间到物理地址空间映射关系的，是一个4KB的页目录
- 另一方面根据页表页的定义：记录这4MB（连续）地址空间到物理地址空间映射关系的，是一个4KB的页表页（当然，它属于整个4MB页表的一部分）
- 所以，页目录和上述页表页内容相同，页目录无需额外分配单独的存储空间
- 页目录就是上述4MB页表中的一页

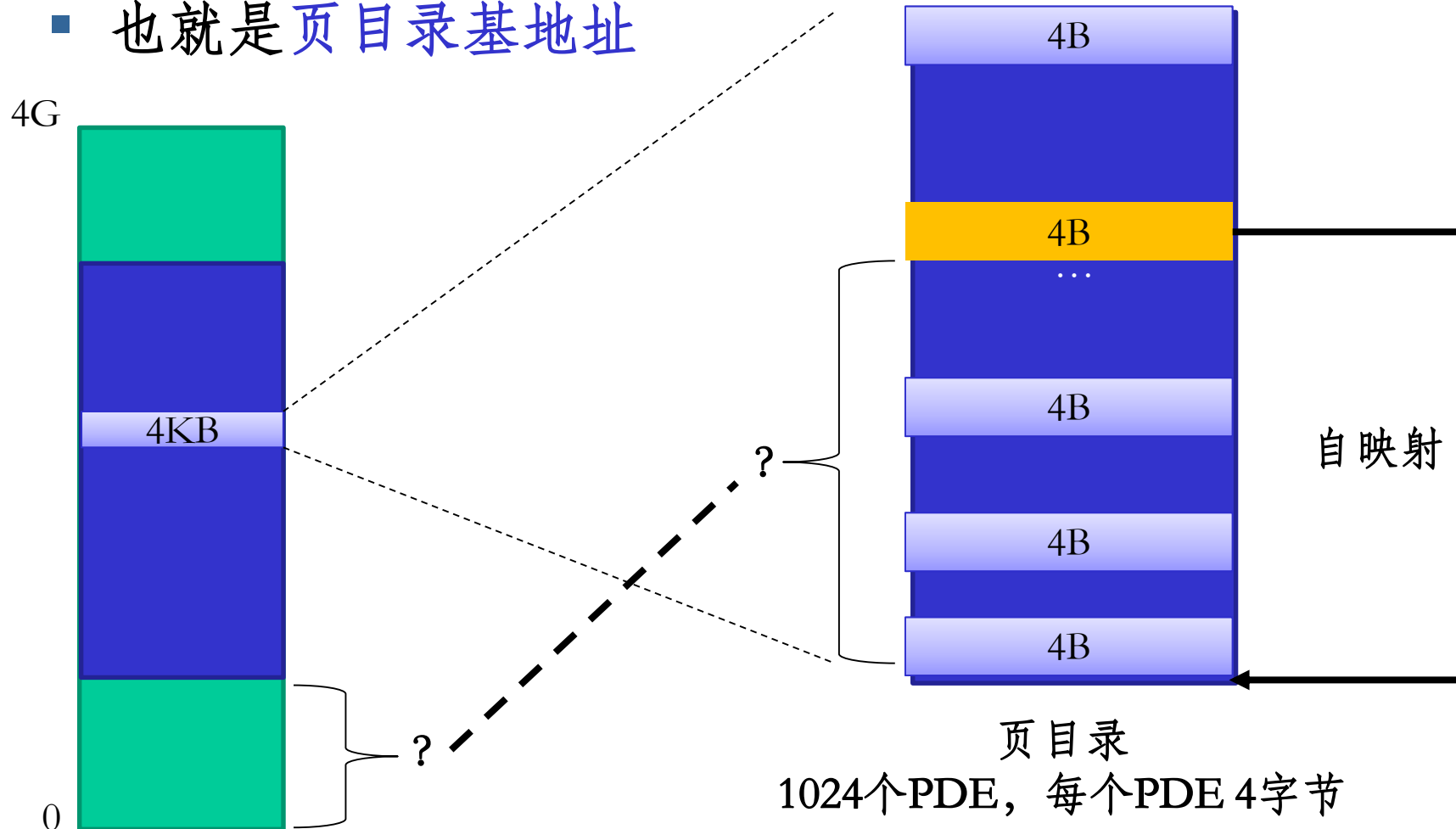
# 页目录自映射





# 页目录自映射

- 自映射：页目录中有一条PDE指向自身物理地址
- 也就是页目录基地址



# 构建方法

1. 给定一个页表基址 $PT_{base}$ ，该基址需4M对齐，即：

$$PT_{base} = ((PT_{base}) \gg 22) \ll 22;$$

不难看出， $PT_{base}$ 的低22位全为0。

2. 页目录表基址 $PD_{base}$ 在哪里？

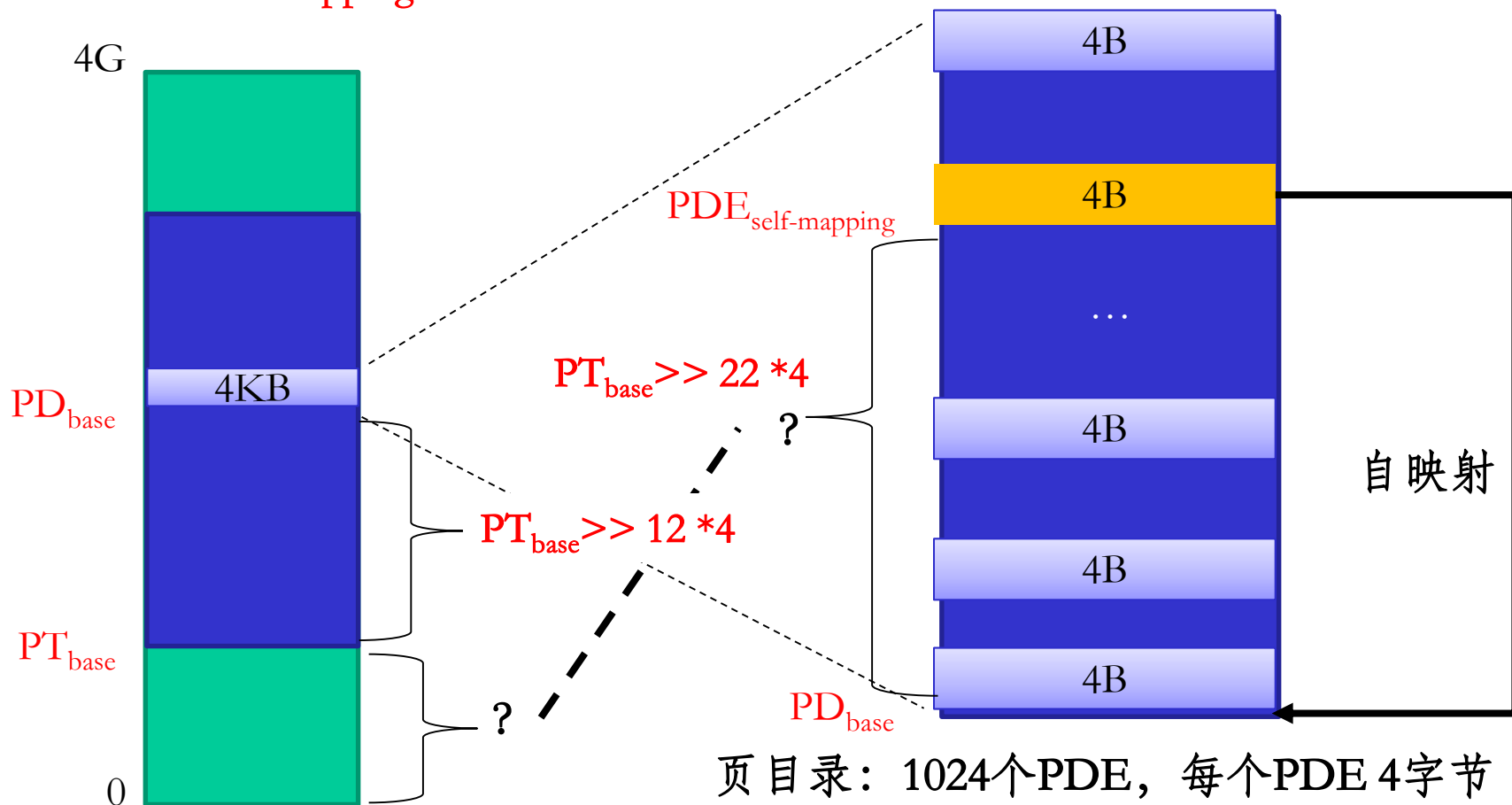
$$PD_{base} = PT_{base} | (PT_{base}) \gg 10$$

3. 自映射目录表项 $PDE_{self-mapping}$ 在哪里？

$$PDE_{self-mapping} = PT_{base} | (PT_{base}) \gg 10 | (PT_{base}) \gg 20$$

# 页目录自映射

- $PD_{base} = PT_{base} \mid (PT_{base}) \gg 10$
- $PDE_{self-mapping} = PT_{base} \mid (PT_{base}) \gg 10 \mid (PT_{base}) \gg 20$

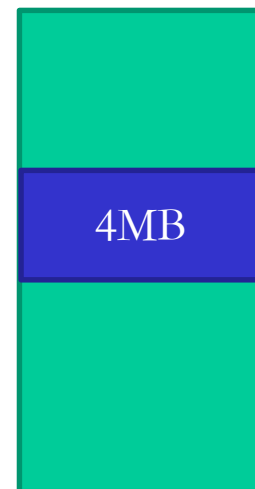


# 思考

4G

$PT_{base}$

0



- 是不是一定要4M对齐?
- 如果仅考虑映射关系，不是必须的。
- 采用4M对齐，可使页目录表单独地存在于一个页面（页表）中，从使用方便性的角度，是必须的。
- 采用4M对齐，还可以简化计算，各部分地址可以采取“拼接”的方式。
- 思考：4MB对齐的地址有什么特点？以下哪个地址是4MB对齐的
  - 0x7fc0 0000, 0x7fd0 0000, 0x8020 0000, 0x1000 0000

# 特别强调

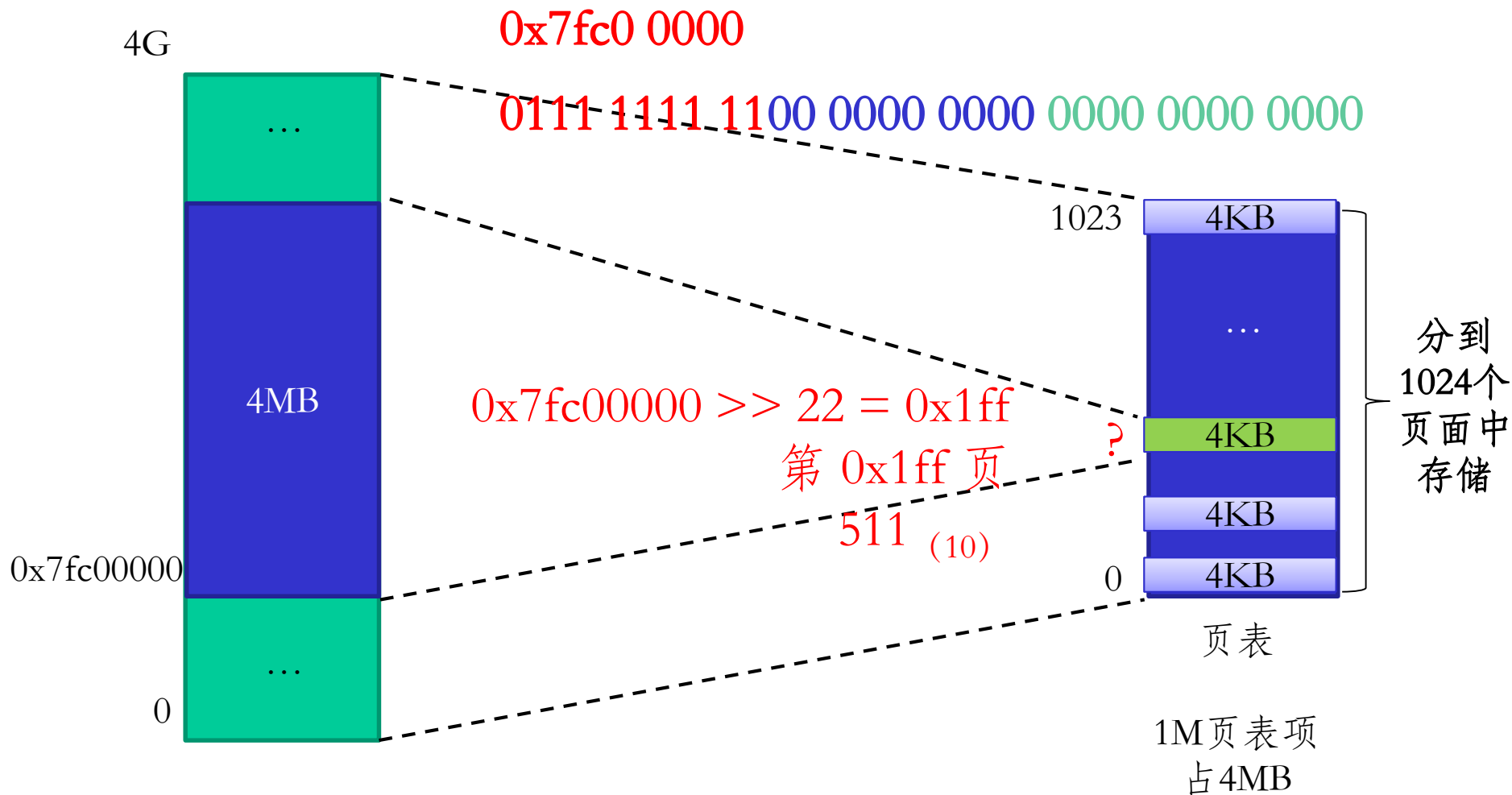
- 只要给定4M对齐的页表基址（虚拟地址），就可以得到所有页表项对应的地址，也就包括页目录表基址和自映射页目录项在页目录表中的位置。因此页目录表基址和自映射页目录项在虚空间中是**计算**出来的。
- 页表主要供OS使用的，因此页表和页目录表通常放置在OS空间中（如Win的高2G空间）；
- “页目录自映射”的含义是页目录包含在页表当中，是我们采用的映射（或组织）方法的一个特征，**是虚拟地址空间内的映射，与虚拟地址到物理地址的映射无关！**
- 支持“页目录自映射”可节省4K（虚拟地址）空间

# 页目录自映射

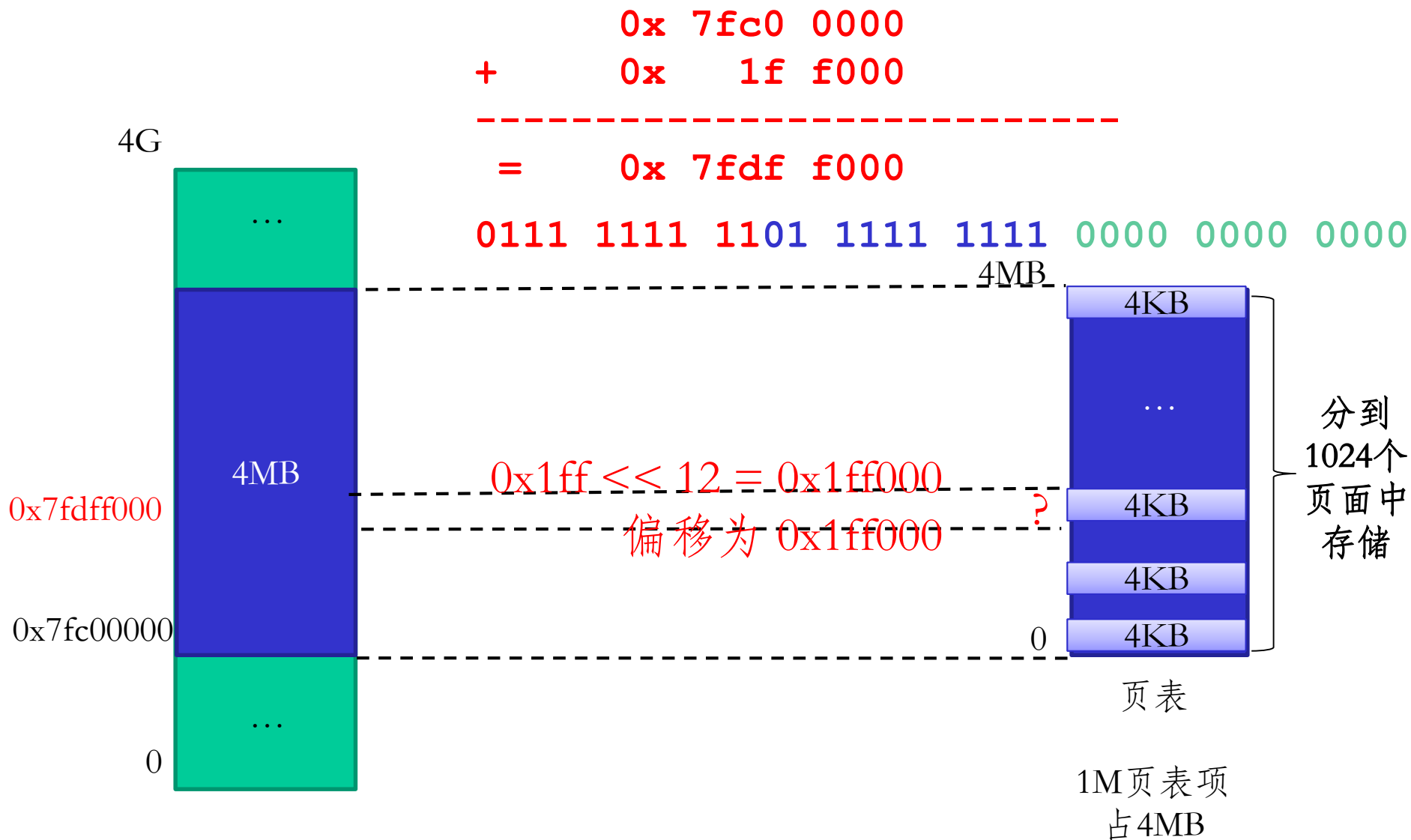
## ■ 举例：页目录在哪？

- 给定页表虚拟地址起始位置，例如0x7fc00000
- 可知，从这个地址开始的4MB是存储页表的空间
- 这4MB地址空间是整个4GB地址空间中第（ $0x7fc00000 \gg 22$ ）个4MB地址空间，因此其逻辑-物理映射关系应该记录在第（ $0x7fc00000 \gg 22$ ）个页表页中

# 页目录自映射



# 页目录自映射



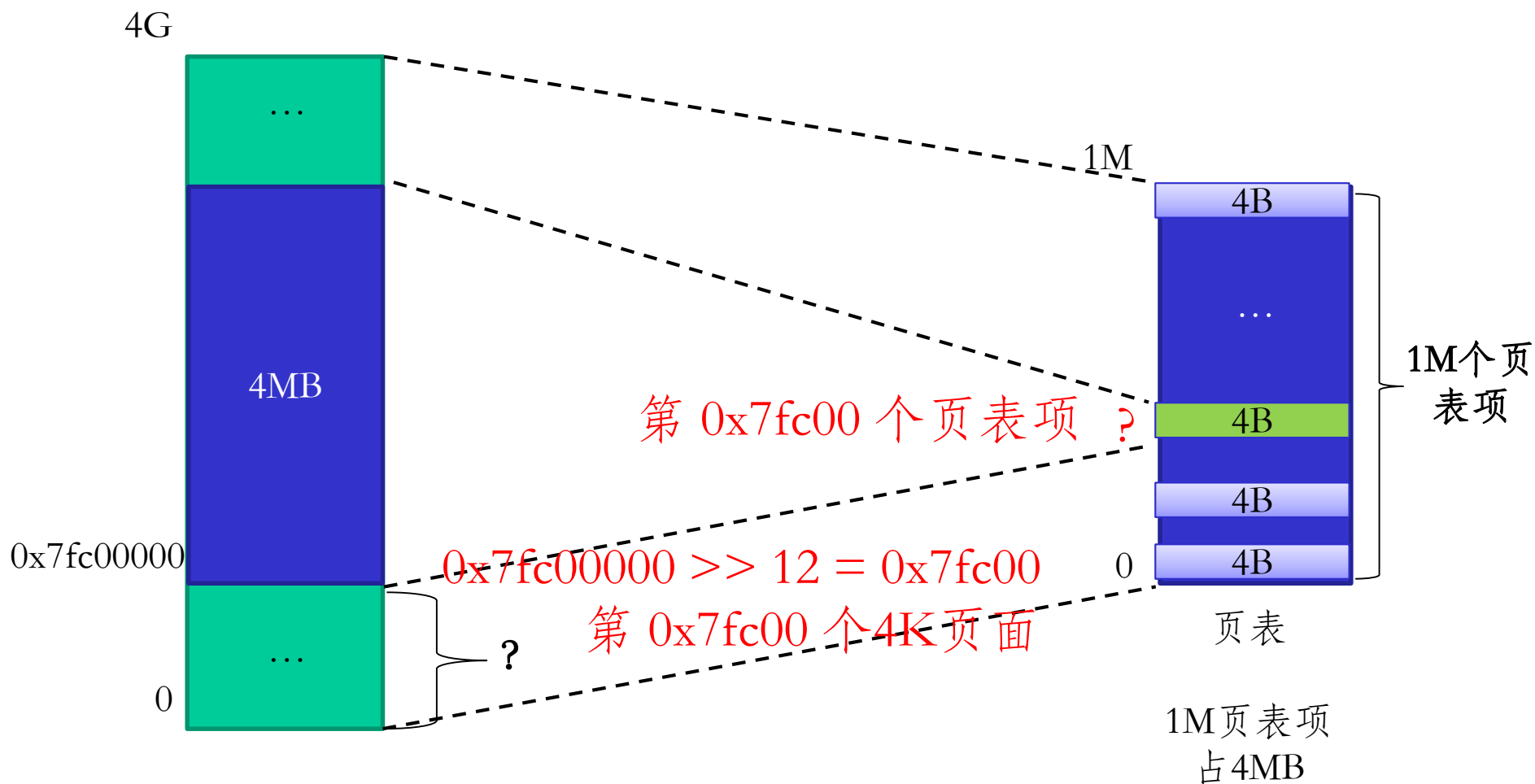
虚拟地址空间



# 页目录自映射

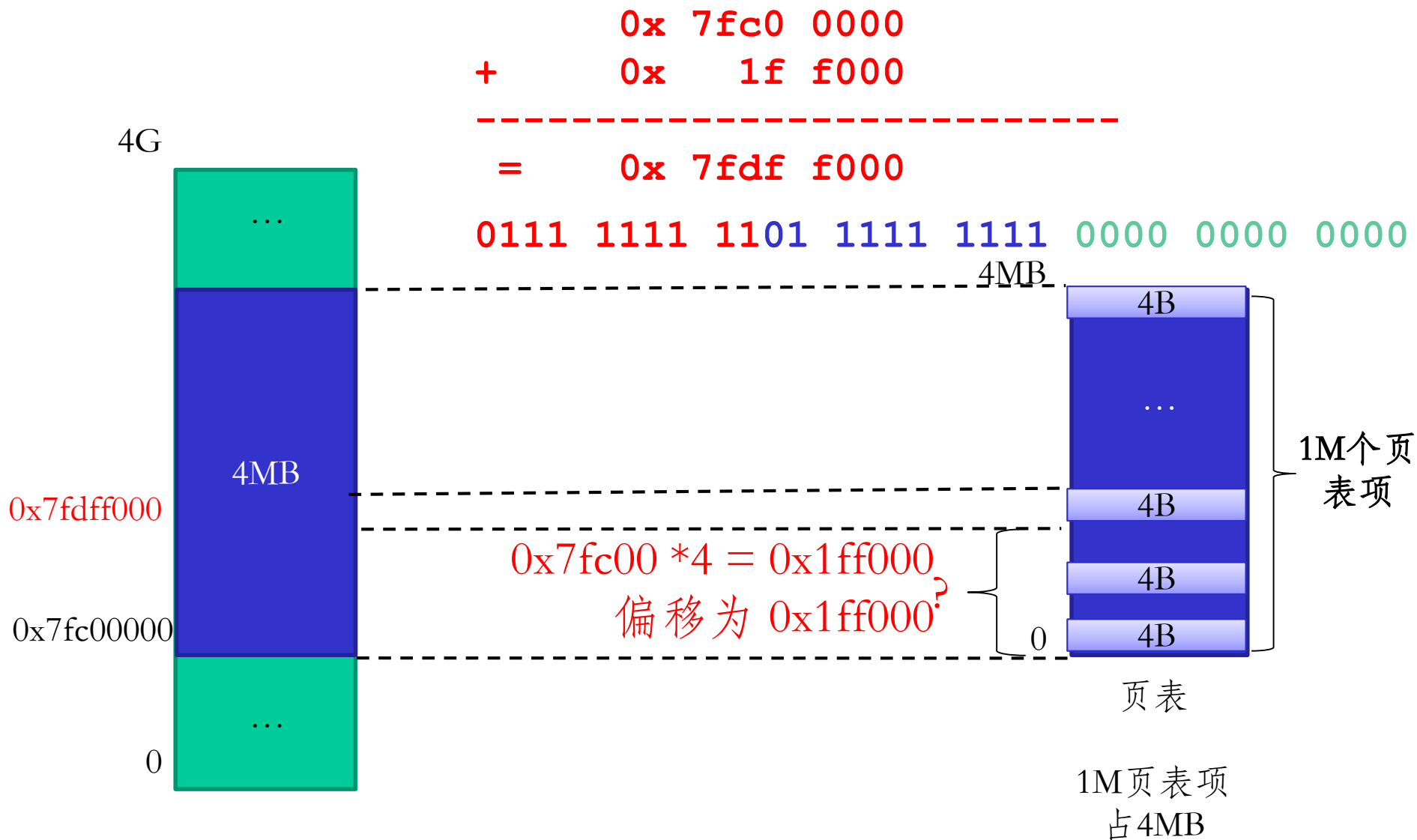
- 页目录在哪？（第二种理解、计算方式）
  - 给定页表虚拟地址起始位置，例如0x7fc00000
  - 将整个4GB地址空间划分为1M个4KB页
  - 上述地址对应于第  $(0x7fc00000 >> 12)$  个4KB页，因此其逻辑-物理映射关系应该记录在第  $(0x7fc00000 >> 12)$  个页表项中
  - 每个页表项4个字节，所以该页表项对于的地址偏移为  $(0x7fc00000 >> 12) * 4 = 0x1ff000$

# 页目录自映射



虚拟地址空间

# 页目录自映射



虚拟地址空间

# 页目录自映射

## ■ 简化计算

- 对于32位地址字长，2级页表，4KB页面大小
- 给定页表起始地址（虚拟地址，**4MB对齐**）b
- 页目录起始地址 =  $b + (b \gg 10) = b + b/1024$

## ■ 练习：

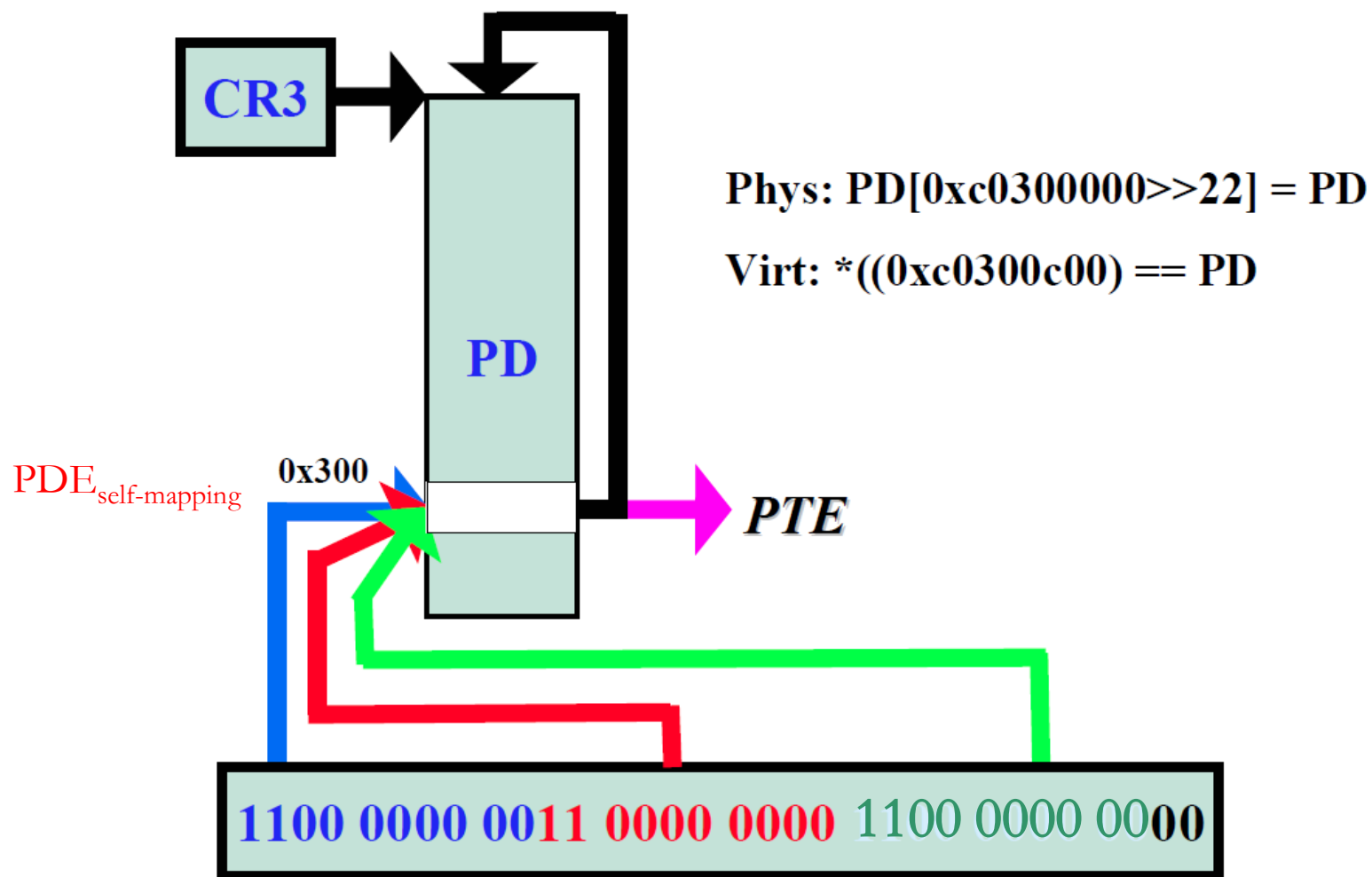
- 页表起始地址0x80000000，页目录起始地址=？
- $0x80000000 + 0x200000 = 0x80200000$

## ■ 反过来：如果给定页目录起始地址，求页表起始地址？

- E.g. 页目录起始地址0xC0300000，页表起始？

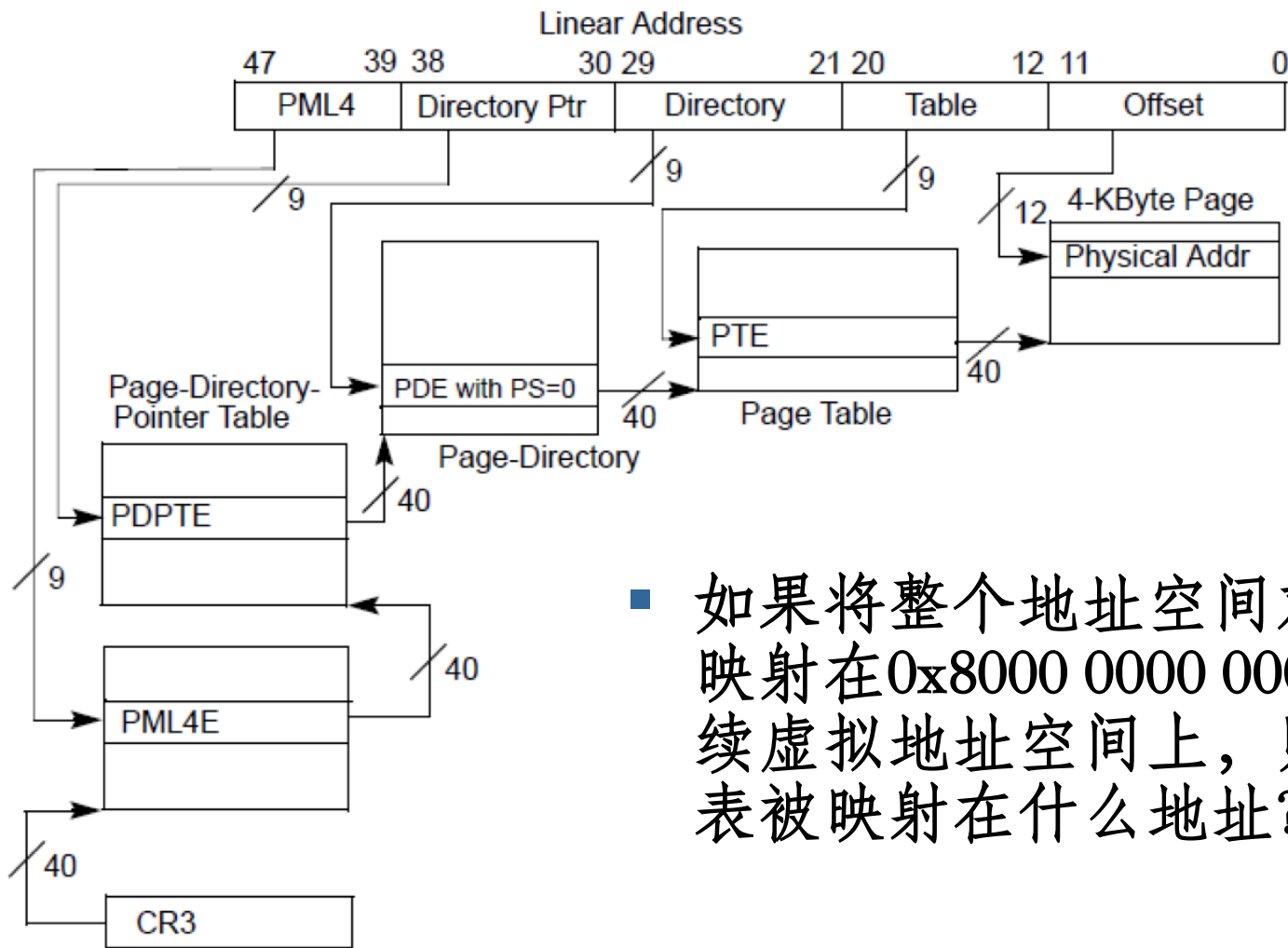
# Win中的页目录自映射

## Virtual Access to PageDirectory[0x300]



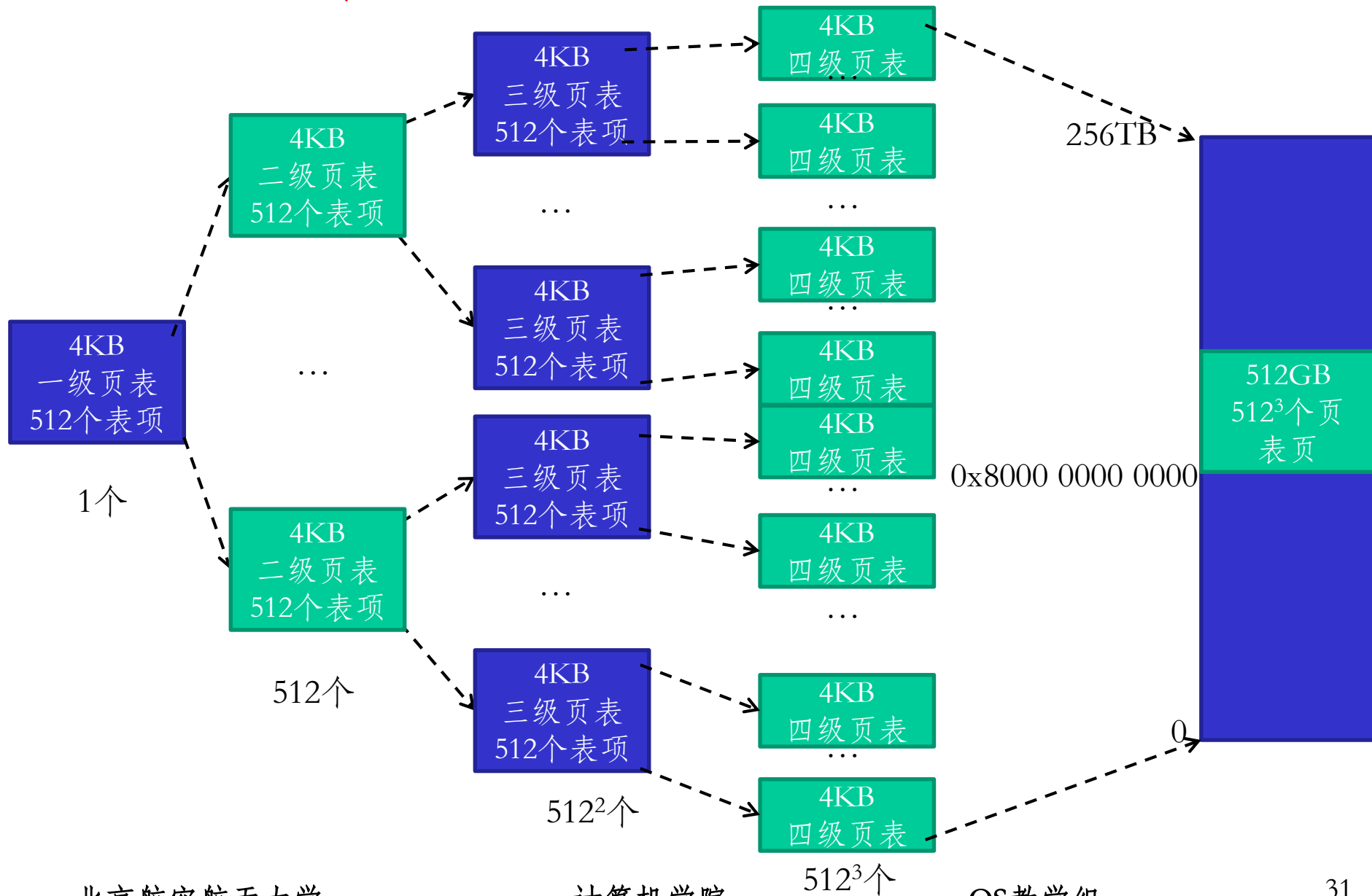
# 思考：推广到更多级页表情况

- 一个48位页式存储系统，采用4级页表：



- 如果将整个地址空间对应的页表映射在0x8000 0000 0000起始的连续虚拟地址空间上，则第一级页表被映射在什么地址？

# 思考：推广到更多级页表情况



# 思考：推广到更多级页表情况

- 整个地址空间大小： $256\text{ TB} = 2^{48}\text{ B}$
- 页大小4KB，每个页表项占8字节
- 页表数量：共有1个一级页表，512个二级页表， $512^2$ 个三级页表， $512^3$ 个四级页表
- $512^3$ 个四级页表映射在整个地址空间中一段连续的512GB地址空间上，起始地址是0x8000 0000 0000
- 实际上， $512^2$ 个三级页表也是连续的，是四级页表的一个子集；512个二级页表也是连续的，是三级页表的一个子集（当然也是整个四级页表的子集）；1个一级页表是上述二级页表的子集（当然最终也是整个四级页表的子集）。也就是说，那个一级页表是整个 $512^3$ 个四级页表中的一个。问题是：是哪个？

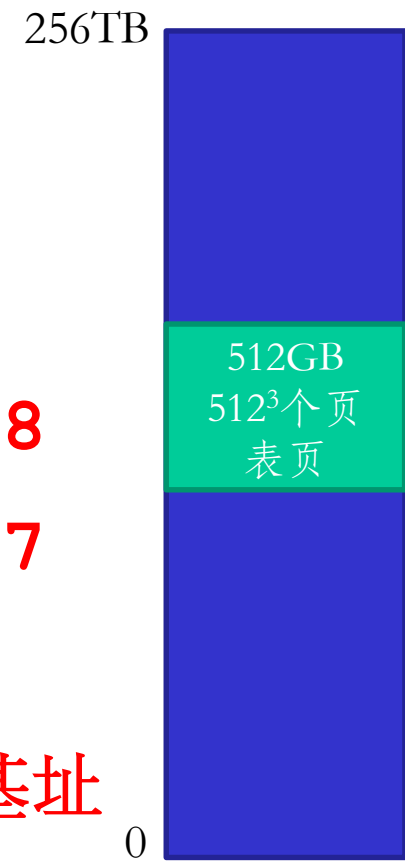


# 思考：推广到更多级页表情况

- 因为：页表映射起始地址是0x8000 0000 0000

$$\begin{array}{rcl} & 0x & 8000 \ 0000 \ 0000 & x \\ + & 0x & \quad 40 \ 0000 \ 0000 & x \gg 9 \\ + & 0x & \quad \quad 2000 \ 0000 & x \gg 18 \\ + & 0x & \quad \quad \quad 10 \ 0000 & x \gg 27 \\ \hline = & 0x & 8040 \ 2010 \ 0000 & \end{array}$$

页目录基址

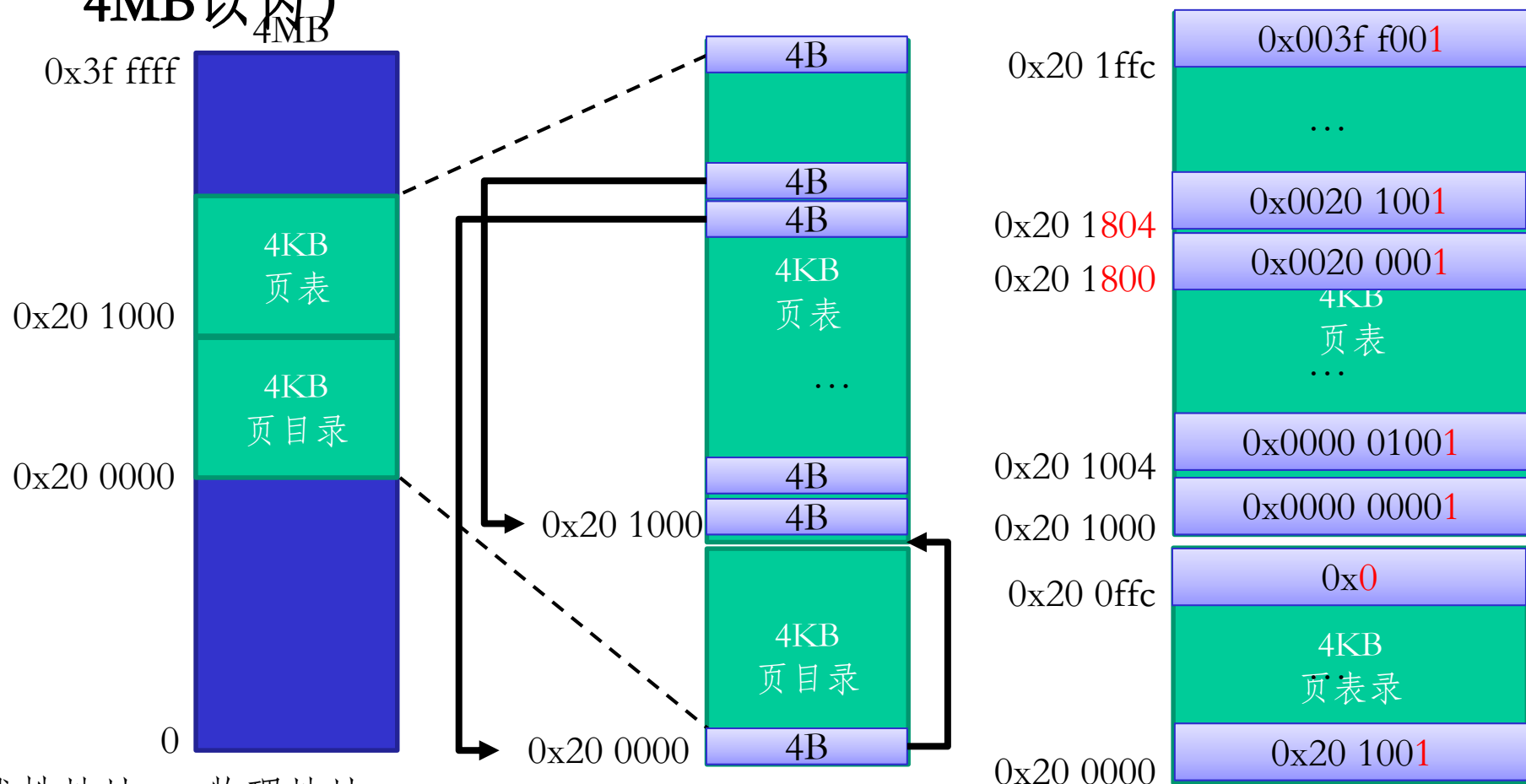


# X86初始系统页表建立

- X86 CPU引导时处于实模式。
- 开启分页（Enable Paging）前，CPU使用物理地址寻址
- 进入保护模式后，可以开启分页，此时CPU将开始使用线性地址寻址。线性地址需要由MMU根据页表翻译成物理地址。
- 问题：怎么实现切换？
- 一个思路：事先构造页表，初始化一部分线性地址空间，使得该空间内的虚拟地址等于物理地址。
  - 例如虚拟地址0x0000 0000～0x0040 0000（4MB）直接映射到物理地址0x0000 0000～0x0040 0000（4MB）
  - 页表怎么构造？

# X86初始系统页表建立

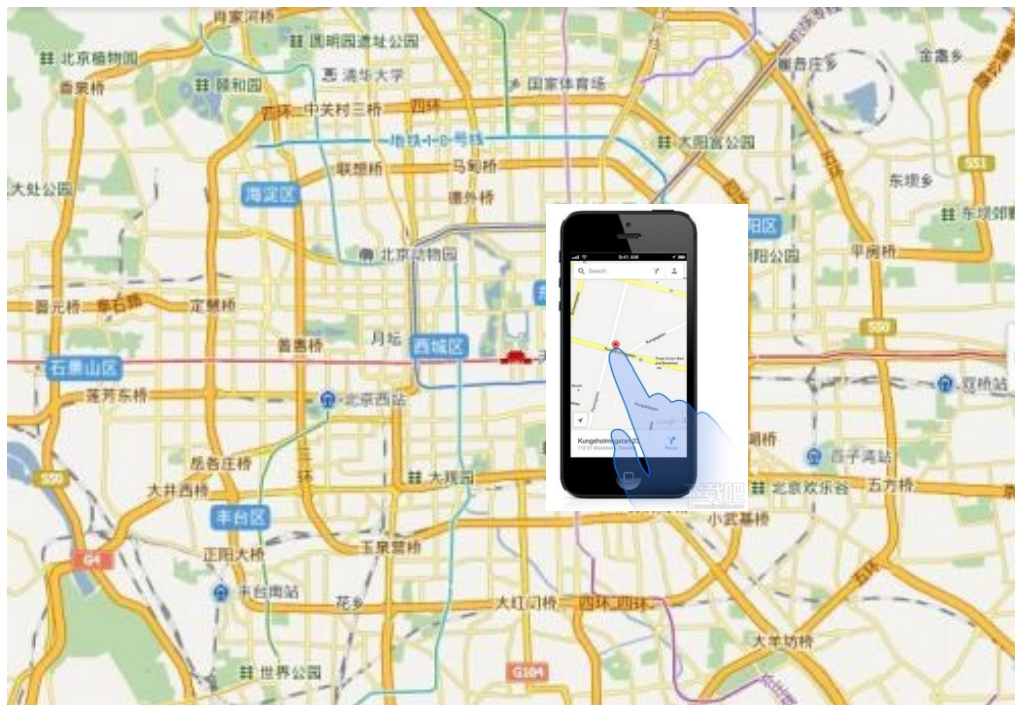
- 在0x20 0000地址处分配2个4K页面，存储页目录和页表。启用分页后，如下页表保证线性地址 == 物理地址（4MB以内）



# 自映射的数学意义

## ■ 地图的比喻

- 手持北京地图在北京
- 必有地图上一点与其表示的地理位置与该点的实际地理位置重合



## ■ 不动点： $f(x) = x$

- 压缩映像



谢谢！

沃天宇

woty@buaa.edu.cn