# 分页存储管理机制

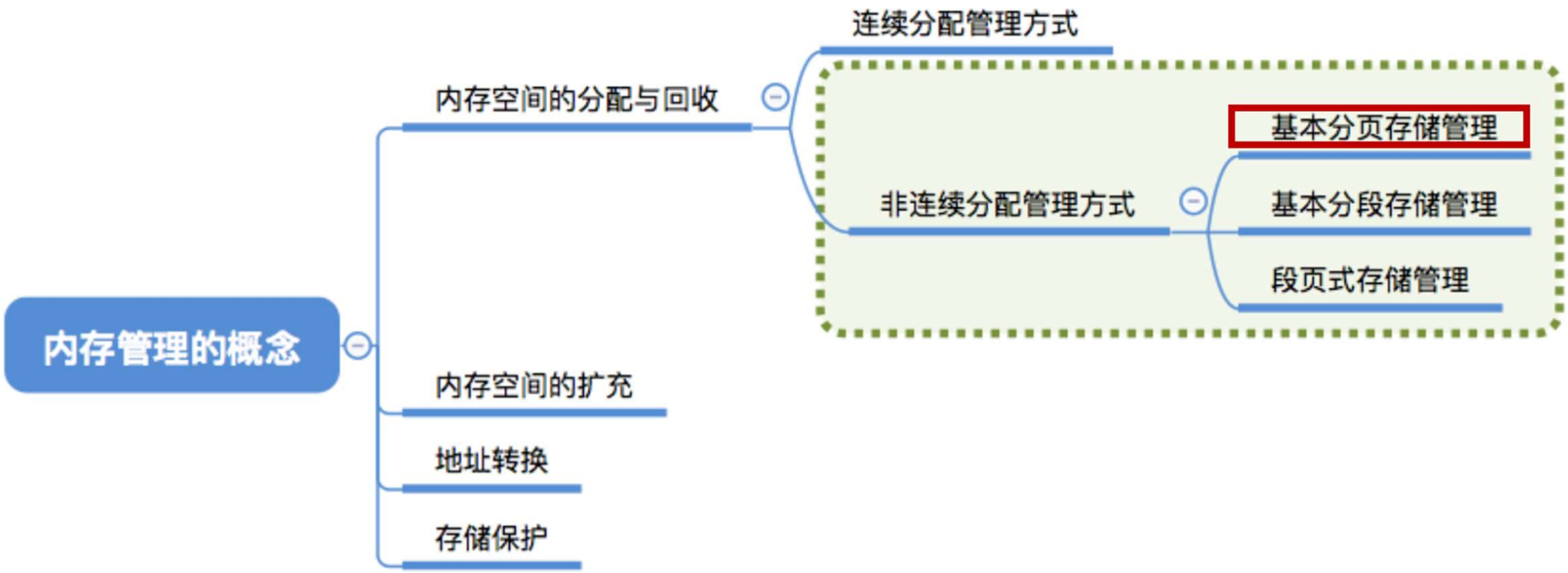
**Linux**
Android
Linux
OpenStack
**Mac OS**
Windows

# 分页存储管理机制

## 本讲内容

1. 逻辑页面与物理页框
2. 分页存储的管理表格
3. 连续存储区管理方案
4. 分区存储的管理方案
5. 物理页框的分配流程

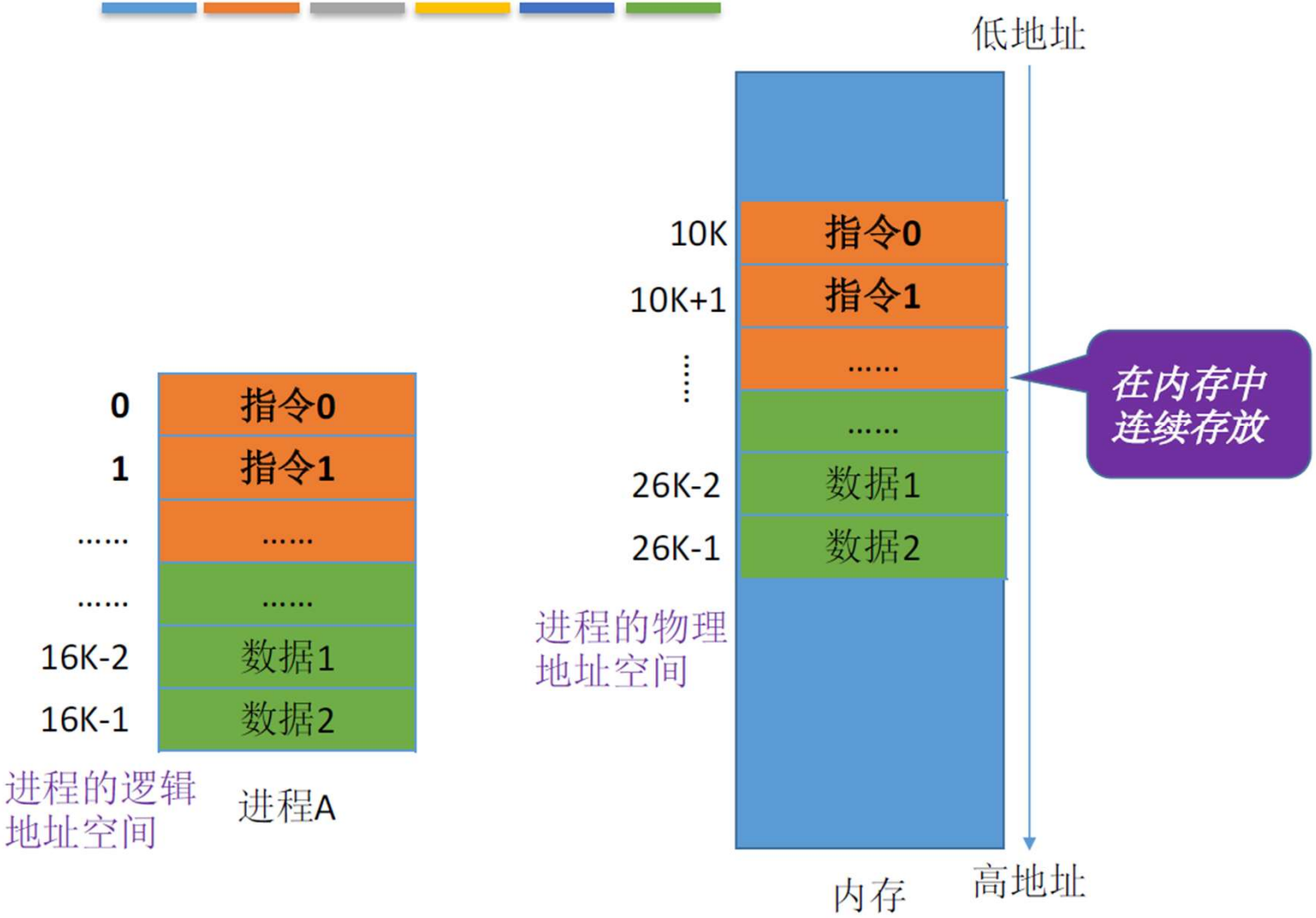
# 知识总览




连续分配：为用户进程分配的必须是一个连续的内存空间。
非连续分配：为用户进程分配的可以是一些分散的内存空间。

# 什么是“地址空间”

概念回顾：
逻辑地址（相对地址）
物理地址（绝对地址）



# 逻辑页面与物理页框

## 1 逻辑页面

15kb进程需要一共4张逻辑页面
4kb 4kb 4kb 3kb 最后一个页面补零的方式补齐

- 进程的逻辑地址空间按固定大小分为若干页面
- 不足一页的补齐为一页，依序从0、1……编号


一共32个bit
高20个bit为逻辑页号，低12个bit为页内地址

# 逻辑页面与物理页框

## 2 物理页框

- 内存空间划分成大小相等的若干存储区
- 每个存储区称为页框，从0开始连续编号

| 页框号 | 页内地址 |
| --- | --- |

一个物理页框正好放下一个逻辑页面

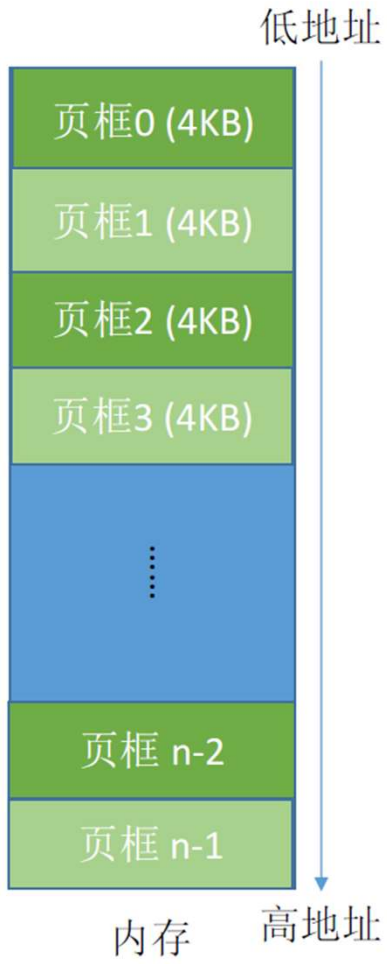
# 分页存储管理机制

## 本讲内容

1. 逻辑页面与物理页框
2. 分页存储的管理表格
3. 分页存储的地址转换
4. 相联存储与快表技术
5. 物理页框的分配流程

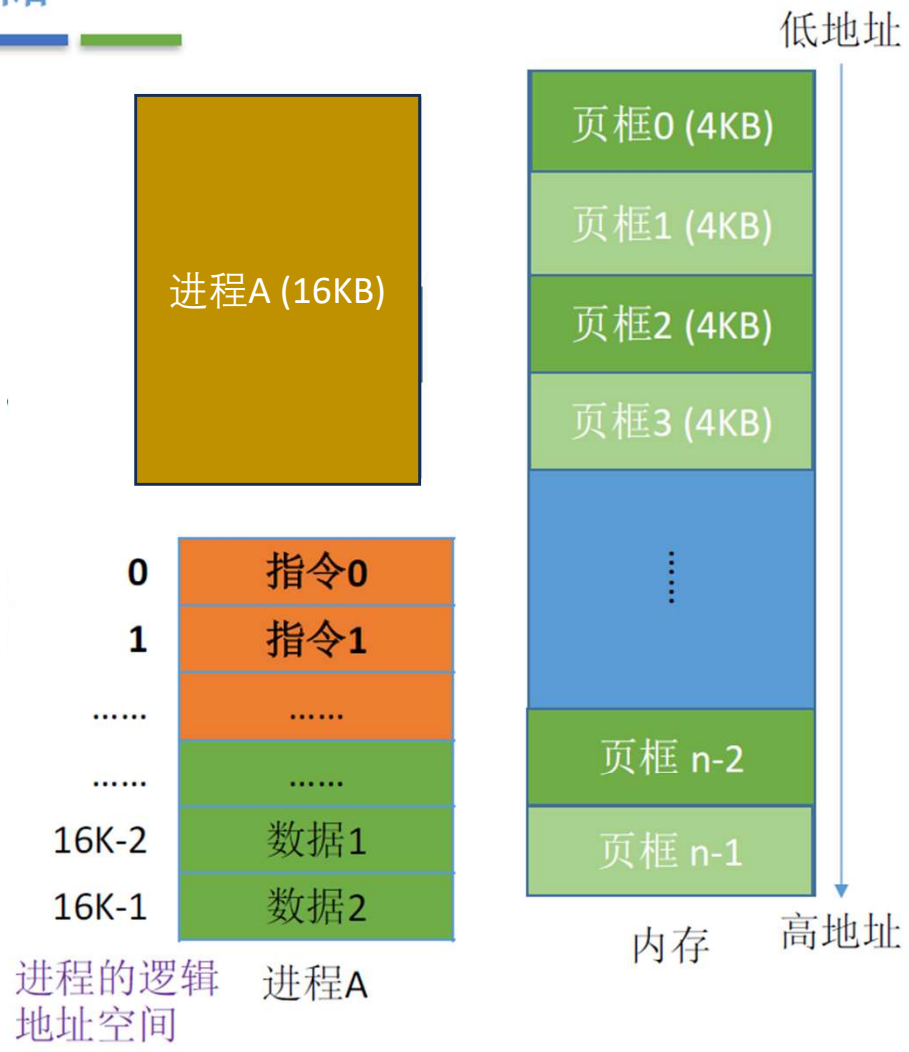
# 什么是分页存储

将内存空间分为一个个大小相等的分区（比如：每个分区4KB），每个分区就是一个“页框”（页框=页帧=内存块=物理块=物理页面）。每个页框有一个编号，即“页框号”（页框号=页帧号=内存块号=物理块号=物理页号），页框号***从0开始***。



# 什么是分页存储

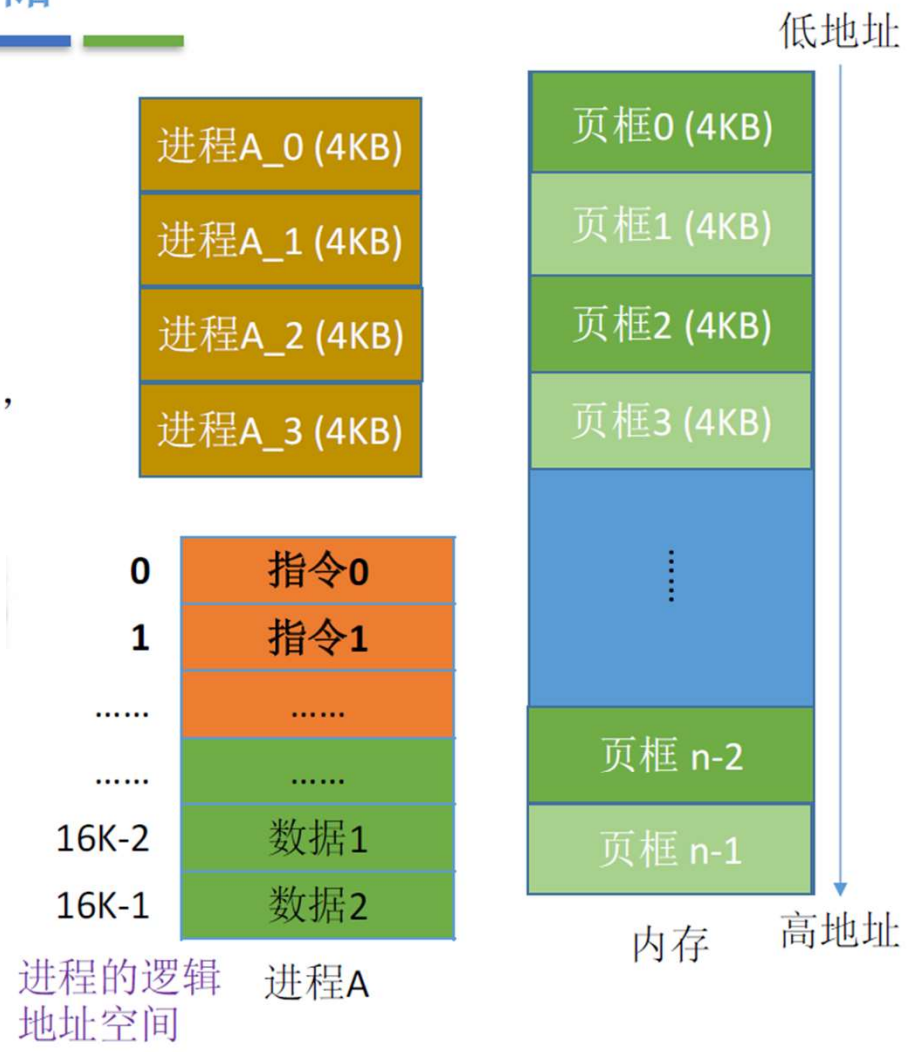
将内存空间分为一个个大小相等的分区（比如：每个分区4KB），每个分区就是一个“页框”（页框=页帧=内存块=物理块=物理页面）。每个页框有一个编号，即“页框号”（页框号=页帧号=内存块号=物理块号=物理页号），页框号***从0开始***。



# 什么是分页存储

将内存空间分为一个个大小相等的分区（比如：每个分区4KB），每个分区就是一个“页框”（页框=页帧=内存块=物理块=物理页面）。每个页框有一个编号，即“页框号”（页框号=页帧号=内存块号=物理块号=物理页号），页框号***从0开始***。

将进程的逻辑地址空间也分为***与页框大小相等***的一个个部分，每个部分称为一个“页”或“页面”。每个页面也有一个编号，即“页号”，页号也是***从0开始***。

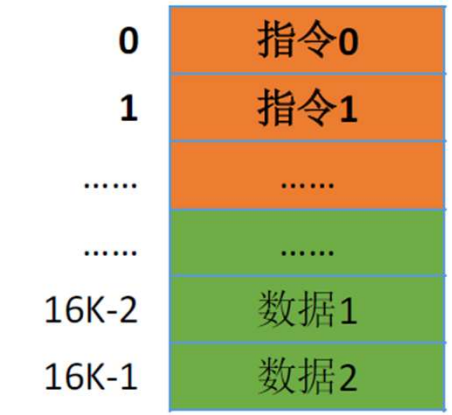


# 什么是分页存储

将内存空间分为一个个大小相等的分区（比如：每个分区4KB），每个分区就是一个“页框”（页框=页帧=内存块=物理块=物理页面）。每个页框有一个编号，即“页框号”（页框号=页帧号=内存块号=物理块号=物理页号），页框号***从0开始***。

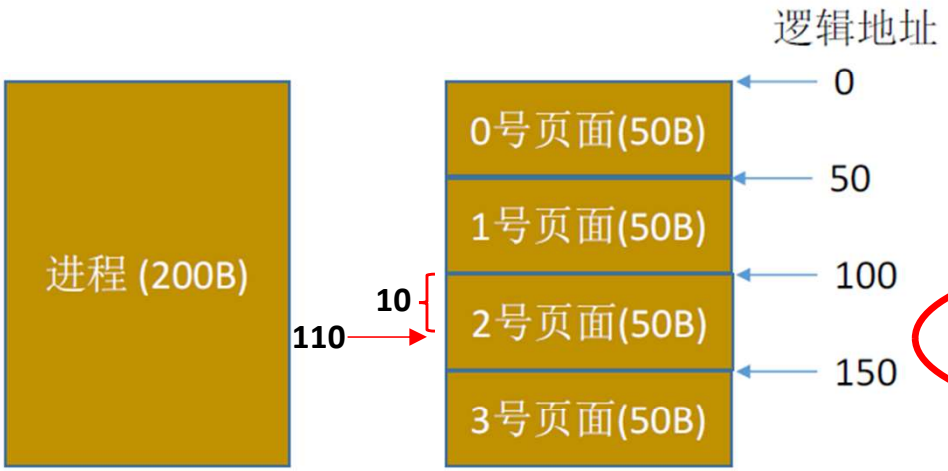
将进程的逻辑地址空间也分为***与页框大小相等***的一个个部分，每个部分称为一个“页”或“页面”。每个页面也有一个编号，即“页号”，页号也是***从0开始***。

操作系统以页框为单位为各个进程分配内存空间。进程的每个页面分别放入一个页框中。也就是说，进程的页面与内存的页框有一一对应的关系。
各个页面不必连续存放，可以放到不相邻的各个页框中。




# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

Eg：在某计算机系统中，页面大小是50B。某进程逻辑地址空间大小为200B，则逻辑地址 110 对应的页号、页内偏移量是多少？




如何计算：

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）

页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

页号 = 110 / 50 = 2

页内偏移量 = 110 % 50 = 10

逻辑地址 可以拆分为（页号，页内偏移量）

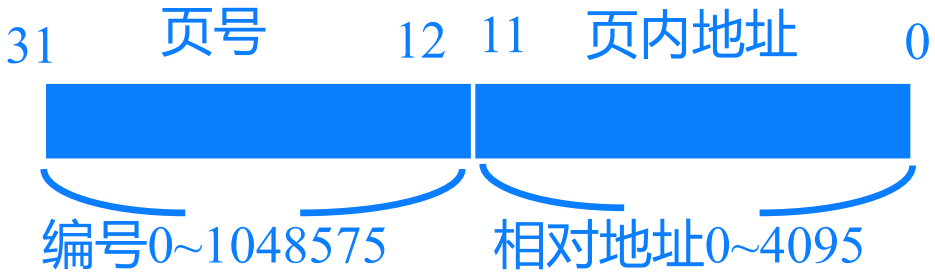
通过页号查询页表，可知页面在内存中的起始地址

页面在内存中的起始地址+页内偏移量 = 实际的物理地址

# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）
页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

假设某计算机用32 个二进制位表示逻辑地址，页面大小为 4KB = $2^{12}$B = 4096B




结论：如果每个页面大小为 $2^K$B，用二进制数表示逻辑地址，则末尾 K 位即为页内偏移量，其余部分就是页号

# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）
页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

假设某计算机用32 个二进制位表示逻辑地址，页面大小为 4KB = $2^{12}$B = 4096B

0号页的逻辑地址范围应该是 0~4095，用二进制表示应该是：
00000000000000000000000000000000 ~ 00000000000000000000111111111111

1号页的逻辑地址范围应该是 4096~8191，用二进制表示应该是：
00000000000000000001000000000000 ~ 00000000000000000001111111111111

2号页的逻辑地址范围应该是 8192~12287，用二进制表示应该是：
00000000000000000010000000000000 ~ 00000000000000000010111111111111

结论：如果每个页面大小为 $2^K$B，用二进制数表示逻辑地址，则末尾 K 位即为页内偏移量，其余部分就是页号

# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）
页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

假设某计算机用32 个二进制位表示逻辑地址，页面大小为 4KB = $2^{12}$B = 4096B

0号页的逻辑地址范围应该是 0~4095，用二进制表示应该是：
00000000000000000000000000000000 ~ 00000000000000000000111111111111

1号页的逻辑地址范围应该是 4096~8191，用二进制表示应该是：
00000000000000000001000000000000 ~ 00000000000000000001111111111111

2号页的逻辑地址范围应该是 8192~12287，用二进制表示应该是：
00000000000000000010000000000000 ~ 00000000000000000010111111111111

Eg：逻辑地址 2，用二进制表示应该是 00000000000000000000000000000010
页号 = 2/4096 = 0 = 00000000000000000000，页内偏移量 = 2%4096 = 2 = 000000000010

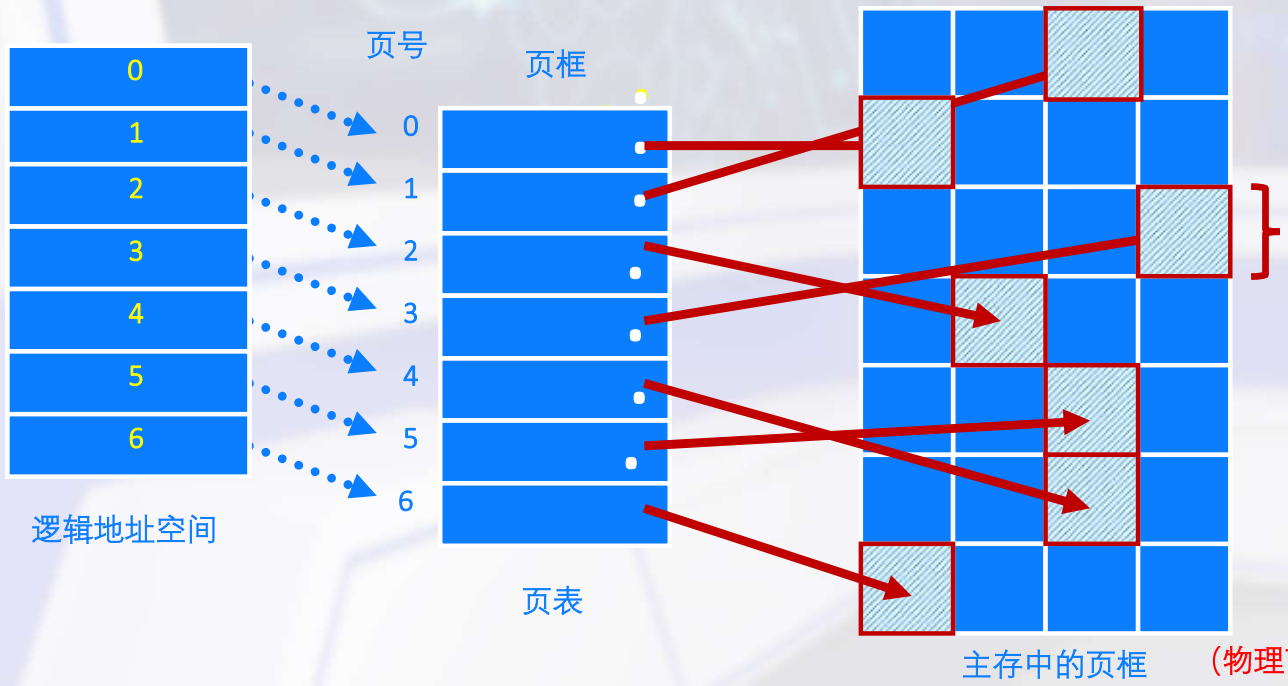
Eg：逻辑地址 4097，用二进制表示应该是 00000000000000000001000000000001
页号 = 4097/4096 = 1 = 00000000000000000001，页内偏移量 = 4097%4096 = 1 = 000000000001

结论：如果每个页面大小为 $2^K$B，用二进制数表示逻辑地址，则末尾 K 位即为页内偏移量，其余部分就是页号

# 分页存储的管理表格

## 1 页表

一共有7个逻辑页面不连续存放
页面内是连续




一个进程就需要一张页表
页表建立逻辑页面和物理页框的对应关系

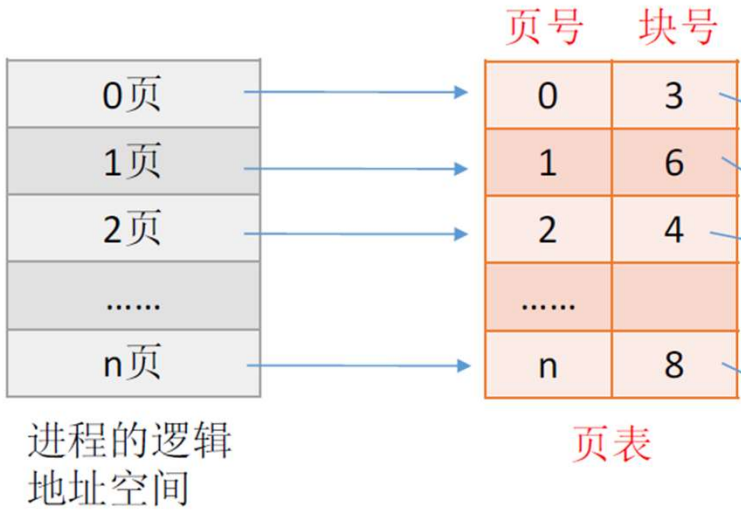
# 分页存储的管理表格

## 1 页表

- 每个进程一张页表，给出页号和页框号对应关系
- 页表放在内存，属于进程的现场信息
- 页表在进程装入内存时，根据内存分配情况建立

# 重要的数据结构——页表

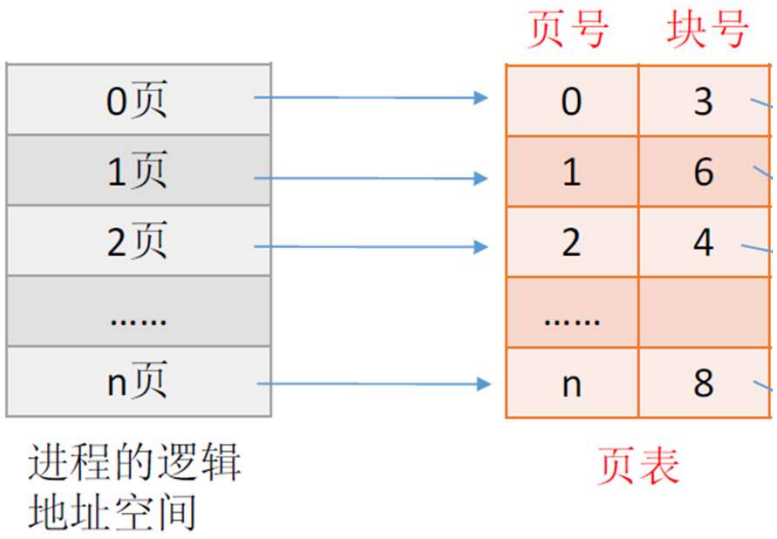
为了能知道进程的每个页面在内存中存放的位置，操作系统要为每个进程建立一张页表。
注：页表通常存在PCB（进程控制块）中

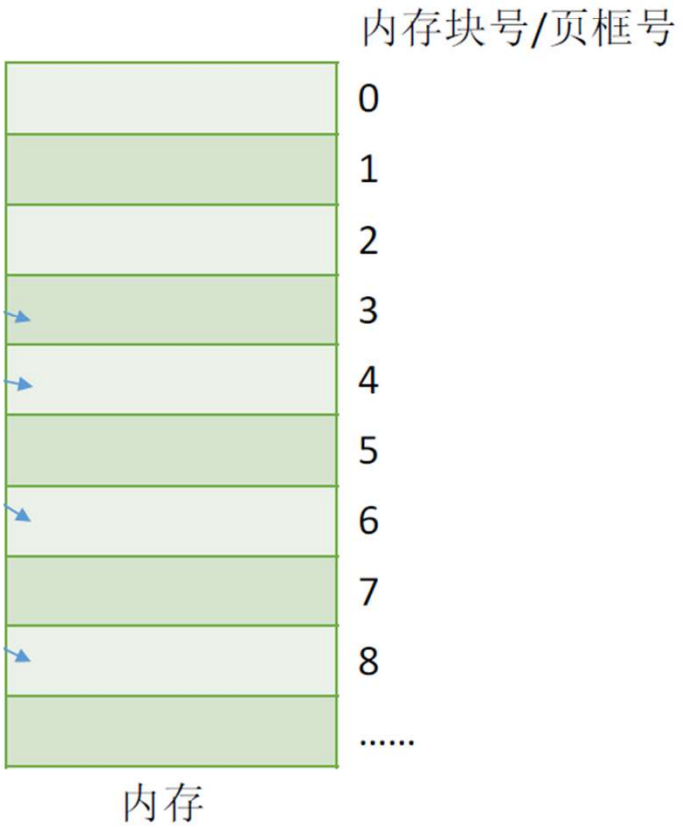



1. 一个进程对应一张页表

# 重要的数据结构——页表

为了能知道进程的每个页面在内存中存放的位置，操作系统要为每个进程建立一张页表。
注：页表通常存在PCB（进程控制块）中



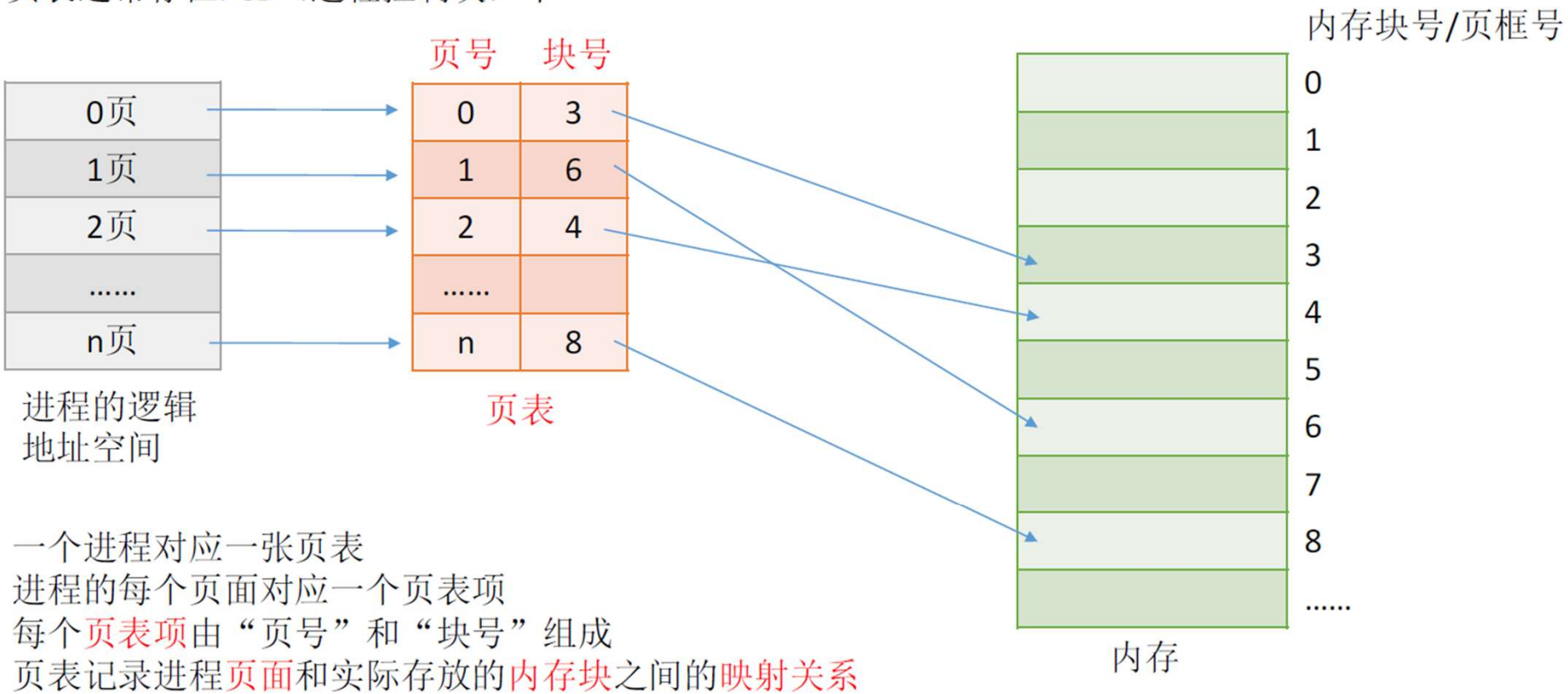




1. 一个进程对应一张页表
2. 进程的每个页面对应一个页表项
3. 每个页表项由“页号”和“块号”组成

# 重要的数据结构——页表

为了能知道进程的每个页面在内存中存放的位置，操作系统要为每个进程建立一张页表。
注：页表通常存在PCB（进程控制块）中

page frame




1. 一个进程对应一张页表
2. 进程的每个页面对应一个页表项
3. 每个页表项由“页号”和“块号”组成
4. 页表记录进程页面和实际存放的内存块之间的映射关系
5. 每个页表项的长度是相同的

# 分页存储的管理表格

## 2 请求表

页表长度=页表行数=进程所需页面总数=最大页号

- 管理每个进程的页表的起始地址和长度
- 请求表整个系统一张

找到每个进程的页表

| 进程号 | 请求页面数 | 页表始址 | 页表长度 | 状态 |
|---|---|---|---|---|
| 1 | 20 | 1024 | 20 | 已分配 |
| 2 | 34 | 1044 | 34 | 已分配 |

# 分页存储的管理表格

## 3 存储页面表

- 整个系统一张，指出内存各物理页框是否已被分配出去，以及空闲页面的总数
- 存储页面表构成方法：位示图和空闲页面链表

# 分页存储的管理表格

## 3 存储页面表

页框 = 内存块 = 物理块 = 物理页面
页框号 = 内存块号 = 物理块号 = 物理页号

- 位示图：在内存中划分一块固定存储区域，每个比特代表一个页框。如果已被分配，则对应比特位置1，否则置0。

16个字节一组来构成

| | 0 | 1 | 2 | 3 | 4 | … | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | … | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | … | 0 | 0 | 1 | 1 | 1 |
| 2 | 0 | 0 | 0 | 1 | 1 | … | 0 | 0 | 0 | 0 | 0 |
| … | | | | | | | | | | | |

第2、3、4、11、14、15、16、17、18等内存块号被占用

记录哪些位置物理空间是空闲的，可以用于分配内存

# 分页存储管理机制

## 本讲内容

1. 逻辑页面与物理页框
2. 分页存储的管理表格
3. 分页存储的地址转换
4. 相联存储与快表技术
5. 物理页框的分配流程

# 问题二：如何实现地址的转换

*进程在内存中连续存放时，操作系统是如何实现逻辑地址到物理地址的转换的？*

重定位寄存器：指明了进程在内存中的起始位置

100

CPU

+

目标逻辑地址：80

相对于起始位置的“偏移量”

内存

物理地址（绝对地址）

| | | |
|---|---|---|
| | | 100 |
| 100 | 指令1：往地址为 80 的存储单元中写入 1 | |
| 101 | 指令2： | |
| … | … | |
| 180 | 1 | |
| 181 | | |
| … | … | |
| 279 | … | 279 |

# 问题二：如何实现地址的转换

*将进程地址空间分页之后，操作系统该如何实现逻辑地址到物理地址的转换？*

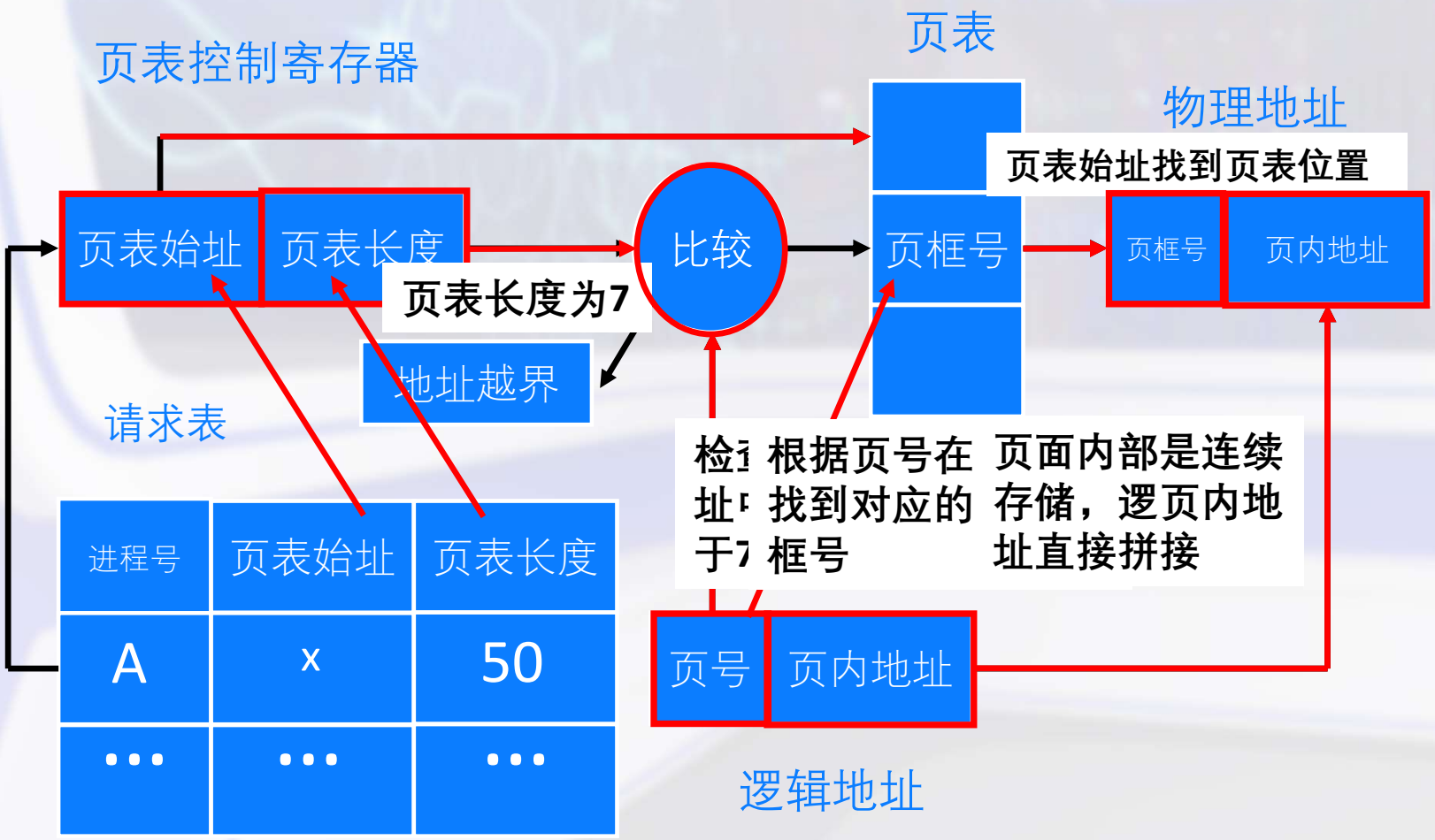


特点：虽然进程的各个页面是离散存放的，但是页面内部是连续存放的

如果要访问逻辑地址 A，则
①确定逻辑地址A 对应的“页号”P
②找到P号页面在内存中的起始地址（需要查页表）
③确定逻辑地址A 的“页内偏移量”W

逻辑地址A 对应的物理地址 = P号页面在内存中的起始地址+页内偏移量W

# 分页存储的地址转换




A 进程 0 1 2 …… 6 页表长度是7

# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）
页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

假设物理地址也用32个二进制位表示，则由于内存块的大小=页面大小，因此：
0号内存块的起始物理地址是 00000000000000000000000000000000
1号内存块的起始物理地址是 00000000000000000001000000000000
2号内存块的起始物理地址是 00000000000000000010000000000000
3号内存块的起始物理地址是 00000000000000000011000000000000

根据页号可以查询页表，而页表中记录的只是内存块号，而不是内存块的起始地址！
J 号内存块的起始地址 = J * 内存块大小

# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）
页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

假设某计算机用32 个二进制位表示逻辑地址，页面大小为 4KB = $2^{12}$B = 4096B

Eg：逻辑地址 4097，用二进制表示应该是 00000000000000000001000000000001
页号 = 4097/4096 = 1 = 00000000000000000001，页内偏移量 = 4097%4096 = 1 = 000000000001

假设物理地址也用32个二进制位表示，则由于内存块的大小=页面大小，因此：
0号内存块的起始物理地址是 00000000000000000000000000000000
1号内存块的起始物理地址是 00000000000000000001000000000000
2号内存块的起始物理地址是 00000000000000000010000000000000
3号内存块的起始物理地址是 00000000000000000011000000000000

根据页号可以查询页表，而页表中记录的只是内存块号，而不是内存块的起始地址！
J 号内存块的起始地址 = J * 内存块大小

# 子问题：如何确定一个逻辑地址对应的页号、页内偏移量？

页号 = 逻辑地址 / 页面长度 （取除法的整数部分）
页内偏移量 = 逻辑地址 % 页面长度（取除法的余数部分）

假设某计算机用32 个二进制位表示逻辑地址，页面大小为 4KB = $2^{12}$B = 4096B

Eg：逻辑地址 4097，用二进制表示应该是 00000000000000000001000000000001
页号 = 4097/4096 = 1 = 00000000000000000001，页内偏移量 = 4097%4096 = 1 = 000000000001

假设物理地址也用32个二进制位表示，则由于内存块的大小=页面大小，因此：
0号内存块的起始物理地址是 00000000000000000000000000000000
1号内存块的起始物理地址是 00000000000000000001000000000000
2号内存块的起始物理地址是 00000000000000000010000000000000
3号内存块的起始物理地址是 00000000000000000011000000000000

假设通过查询页表得知1号页面存放的内存块号是9（1001），则
9号内存块的起始地址 = 9*4096 = 00000000000000001001000000000000
则逻辑地址4097对应的物理地址 = 页面在内存中存放的起始地址 + 页内偏移量
=（00000000000000001001000000000001）

结论：如果页面大小刚好是2的整数幂，则只需把页表中记录的物理块号拼接上页内偏移量就能得到对应的物理地址

# 逻辑地址结构

分页存储管理的逻辑地址结构如下所示：

| 31 ..... 12 | 11 ..... 0 |
| --- | --- |
| 页号 P | 页内偏移量 W |

地址结构包含两个部分：前一部分为页号，后一部分为页内偏移量 W。在上图所示的例子中，地址长度为 32 位，其中 0~11位 为“页内偏移量”，或称“页内地址”；12~31 位为“页号”。

如果有 K 位表示“页内偏移量”，则说明该系统中一个页面的大小是 $2^K$ 个内存单元
如果有 M 位表示“页号”，则说明在该系统中，一个进程最多允许有 $2^M$ 个页面

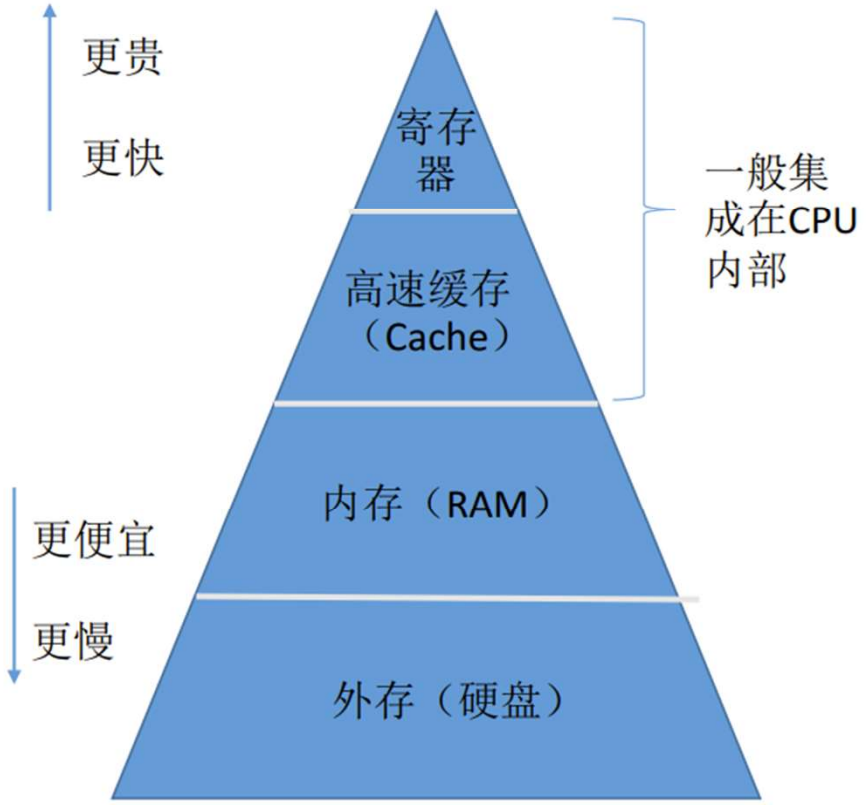
重要重要重要！！！
页面大小 ↔ 页内偏移量位数
➔逻辑地址结构

# 分页存储管理机制

## 本讲内容

1. 逻辑页面与物理页框
2. 分页存储的管理表格
3. 分页存储的地址转换
4. 相联存储与快表技术
5. 物理页框的分配流程

# 什么是快表（TLB）





¥289.00

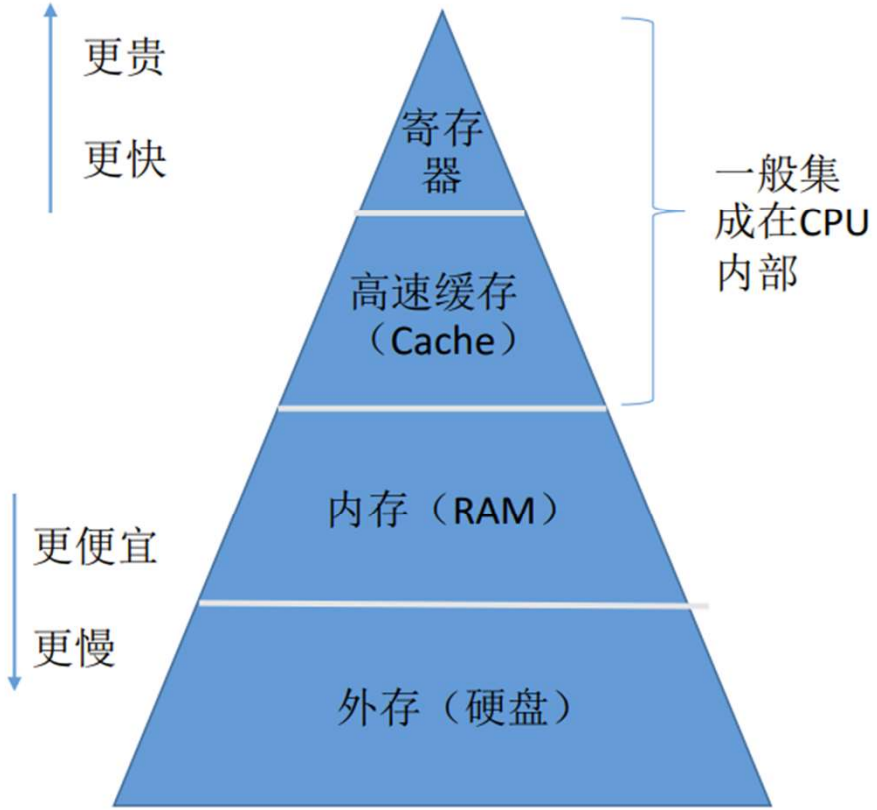
西部数据(WD)蓝盘 1TB SATA6Gb/s 7200


¥299.00

金士顿(Kingston) DDR4 2666 8GB 台式机内存 骇客神条 Fury雷电系列 金士顿.

# 什么是快表（TLB）

快表，又称联想寄存器（TLB， translation lookaside buffer ），是一种访问速度比内存快很多的高速缓存（TLB不是内存！），用来存放最近访问的页表项的副本，可以加速地址变换的速度。与此对应，内存中的页表常称为慢表。





¥289.00

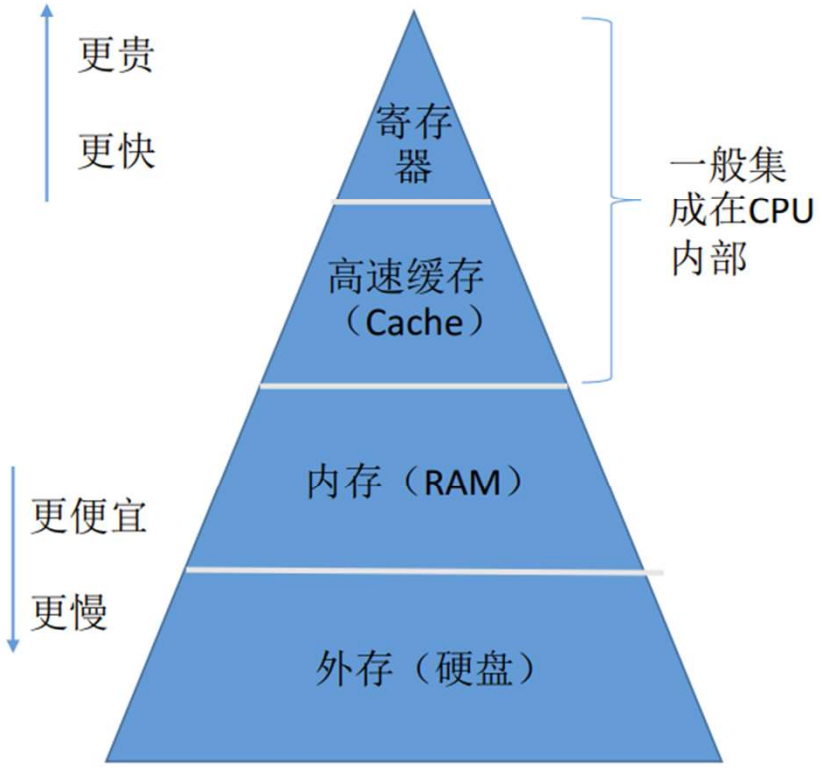
西部数据(WD)蓝盘 1TB SATA6Gb/s 7200


¥299.00

金士顿(Kingston) DDR4 2666 8GB 台式机内存 骇客神条 Fury雷电系列 金士顿.

# 什么是快表（TLB）

快表，又称联想寄存器（TLB， translation lookaside buffer），是一种访问速度比内存快很多的高速缓存（TLB不是内存！），用来存放最近访问的页表项的副本，可以加速地址变换的速度。与此对应，内存中的页表常称为慢表。





￥109.00 包邮 807人付款

简易书架简约现代置物架落地桌上柜子学生创意格子柜自由组合书柜

最近要用到的书放在小书包里


￥345.30 包邮 0人付款

小猪佩奇儿童幼儿园书包女小学生轻

# 相联存储与快表技术

## 1 相联存储与快表

- 相联存储是在MMU(存储管理单元)中设置的专用的高速缓冲存储器
- 快表是相联存储器中存放的最近访问的部分页表
- 最近访问过的内容，将来再次被访问到的概率依然很高(程序的局部性原理)
- 在基本地址变换的基础上，引入快表可以让地址转换的过程更快

# 局部性原理

```
int i = 0;
int a[100];
while (i < 100) {
  a[i] = i;
  i++;
}
```

这个程序执行时，会很频繁地访问 10 号页面、23号页面

时间局部性：如果执行了程序中的某条指令，那么不久后这条指令很有可能再次执行；如果某个数据被访问过，不久之后该数据很可能再次被访问。（因为程序中存在大量的循环）

# 局部性原理

```
int i = 0;
int a[100];
while (i < 100) {
  a[i] = i;
  i++;
}
```

这个程序执行时，会很频繁地访问 10 号页面、23号页面

时间局部性：如果执行了程序中的某条指令，那么不久后这条指令很有可能再次执行；如果某个数据被访问过，不久之后该数据很可能再次被访问。（因为程序中存在大量的循环）

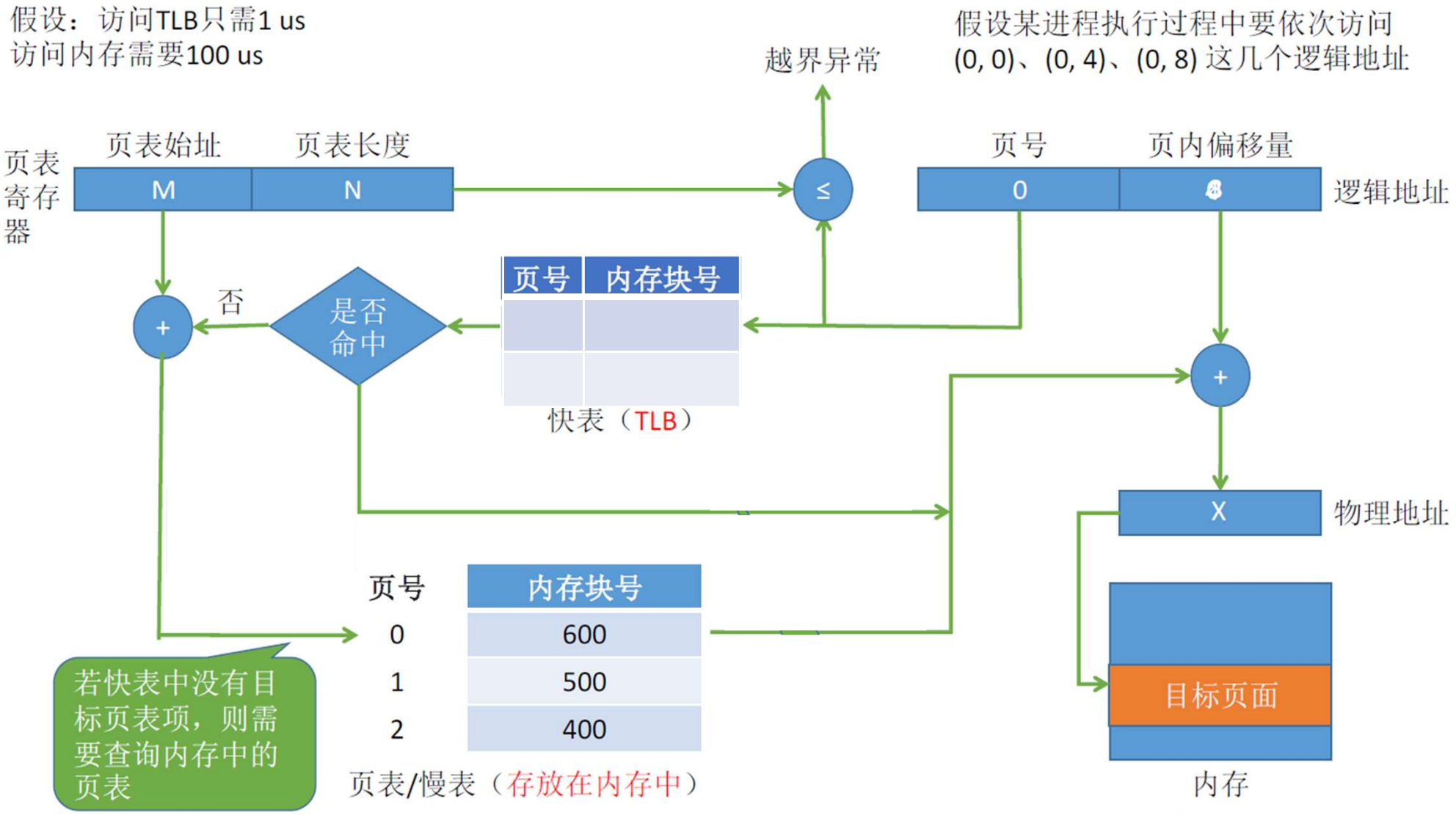
空间局部性：一旦程序访问了某个存储单元，在不久之后，其附近的存储单元也很有可能被访问。（因为很多数据在内存中都是连续存放的）

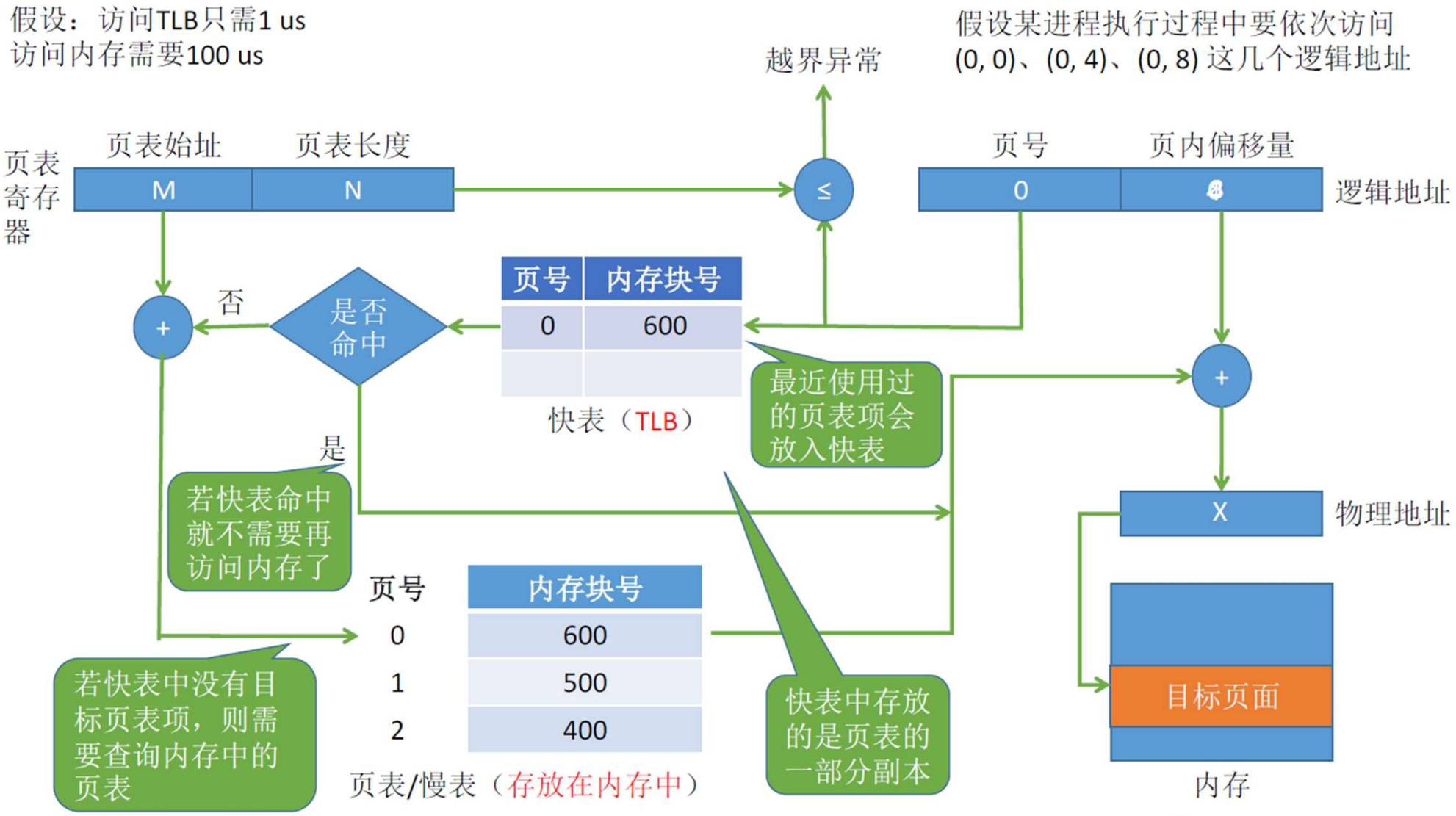
上小节介绍的基本地址变换机构中，每次要访问一个逻辑地址，都需要查询内存中的页表。由于局部性原理，可能连续很多次查到的都是同一个页表项

# 相联存储与快表技术

## 2 地址转换

- 地址变换机构自动将页号与快表中的页号比较
- 快表存在所要访问的页表项，直接读出对应的页框号
- 快表中未找到对应的页表项，则再访问内存中的页表；同时将此页表项存入快表中，修改快表
- 若快表已满，则系统需换出某些页表项
- CPU直接去高速缓存中访问，实现对内存和高速缓冲的联合使用


假设：访问TLB只需1 us
访问内存需要100 us
假设某进程执行过程中要依次访问
(0, 0)、(0, 4)、(0, 8) 这几个逻辑地址
越界异常
页表寄存器
页表始址
页表长度
M
N
≤
页号
页内偏移量
0
逻辑地址
+
否
是否命中
页号
内存块号
快表（TLB）
+
X
物理地址
页号
内存块号
0
600
1
500
2
400
若快表中没有目标页表项，则需要查询内存中的页表
目标页面
页表/慢表（存放在内存中）
内存


假设：访问TLB只需1 us
访问内存需要100 us
假设某进程执行过程中要依次访问
(0, 0)、(0, 4)、(0, 8) 这几个逻辑地址
越界异常
页表寄存器
页表始址
页表长度
M
N
≤
页号
页内偏移量
0
逻辑地址
+
否
是否命中
页号
内存块号
0
600
快表（TLB）
最近使用过的页表项会放入快表
+
是
若快表命中就不需要再访问内存了
X
物理地址
页号
内存块号
0
600
1
500
2
400
若快表中没有目标页表项，则需要查询内存中的页表
快表中存放的是页表的一部分副本
目标页面
内存
页表/慢表（存放在内存中）

# 引入快表后，地址的变换过程

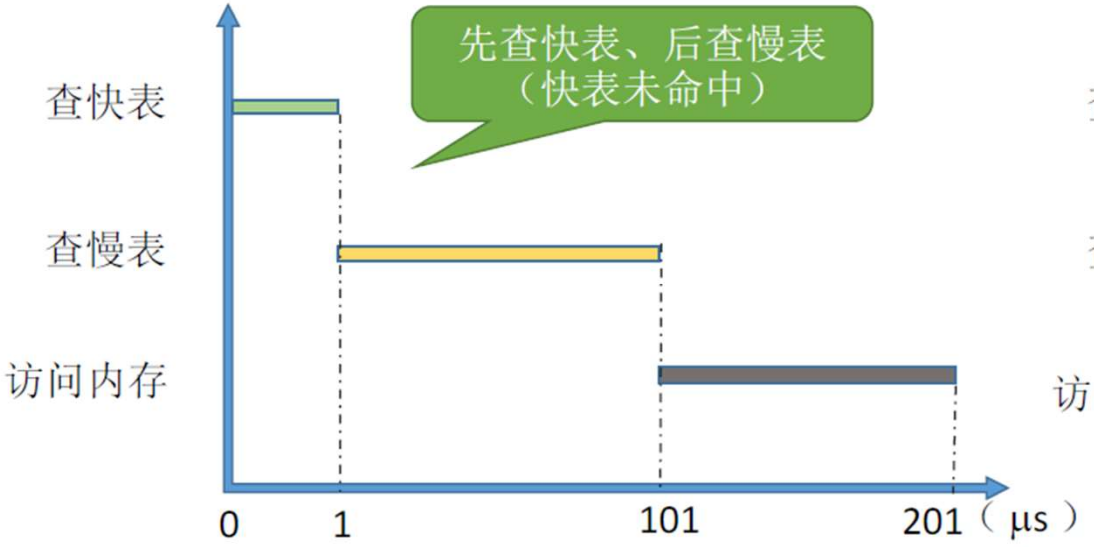
例：某系统使用基本分页存储管理，并采用了具有快表的地址变换机构。访问一次快表耗时 1us，访问一次内存耗时 100us。若快表的命中率为 90%，那么访问一个逻辑地址的平均耗时是多少？

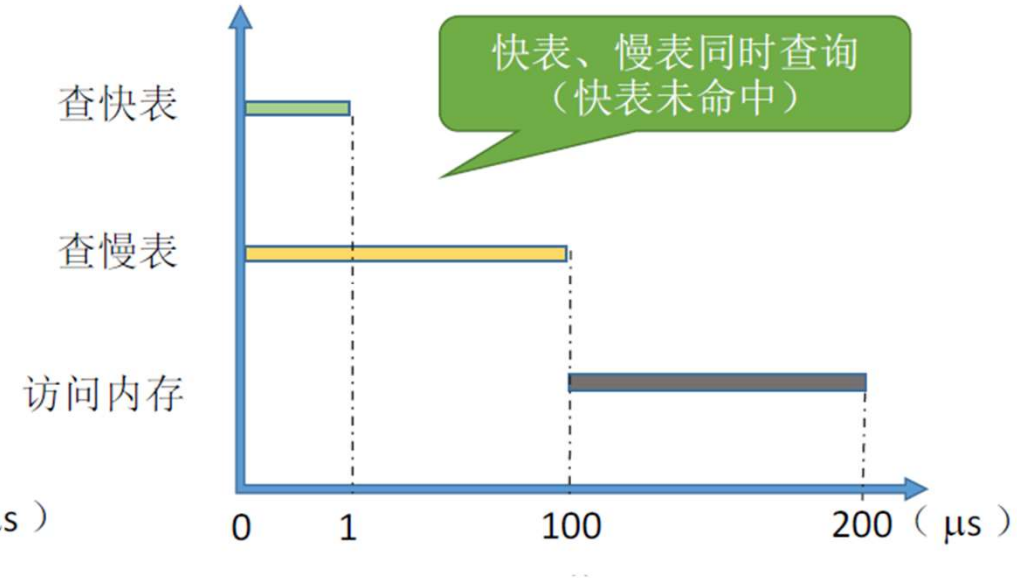
(1+100) * 0.9 + (1+100+100) * 0.1 = 111 us

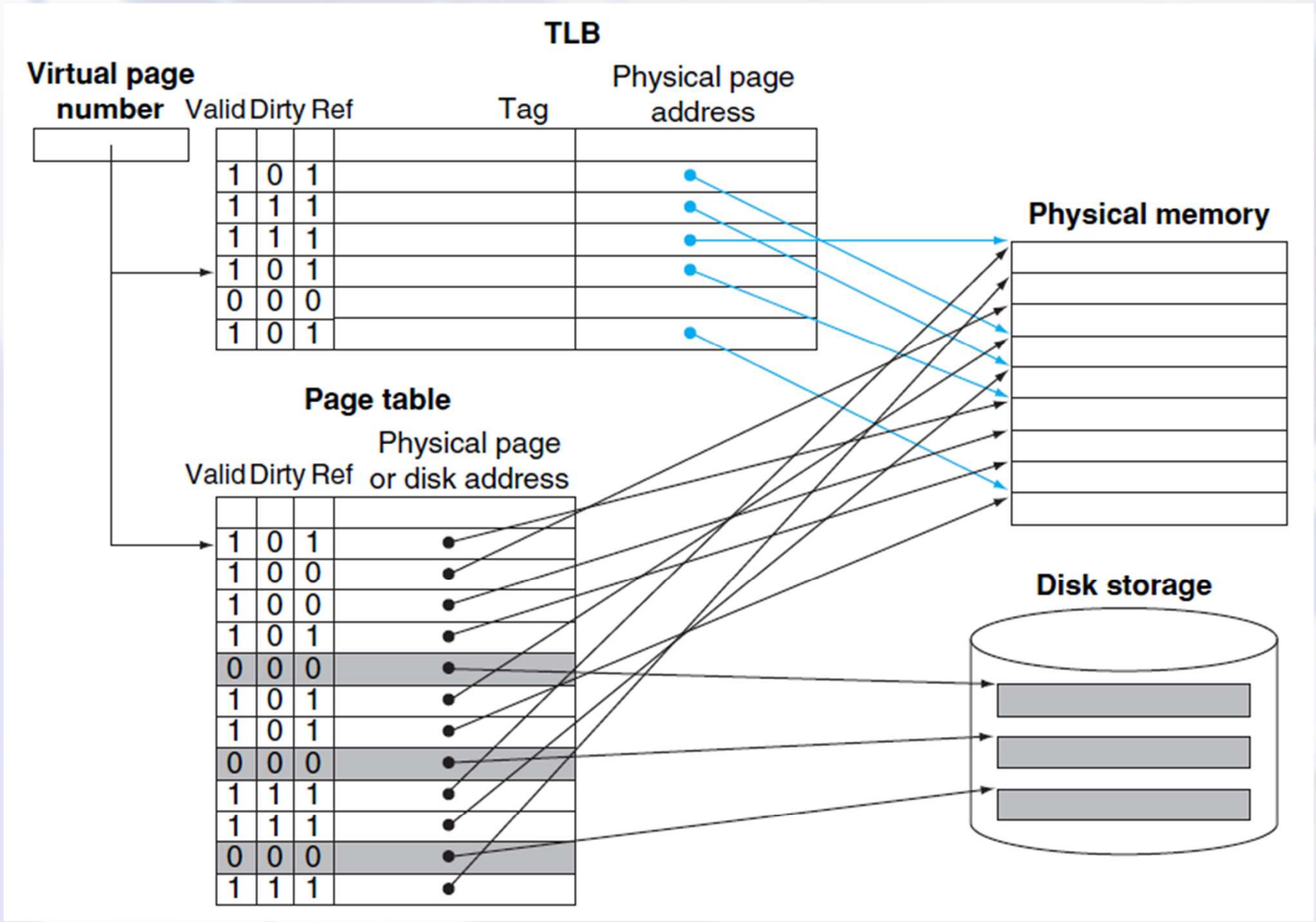
有的系统支持快表和慢表同时查找，如果是这样，平均耗时应该是 (1+100) * 0.9 + (100+100) * 0.1 = 110.9 us

若未采用快表机制，则访问一个逻辑地址需要 100+100 = 200us

显然，引入快表机制后，访问一个逻辑地址的速度快多了。




TLB
Virtual page number
Valid
Dirty
Ref
Tag
Physical page address
1 0 1
1 1 1
1 1 1
1 0 1
0 0 0
1 0 1
Physical memory
Page table
Physical page or disk address
Valid
Dirty
Ref
1 0 1
1 0 0
1 0 0
1 0 1
0 0 0
1 0 1
1 0 1
0 0 0
1 1 1
1 1 1
0 0 0
1 1 1
Disk storage

# 知识回顾与重要考点

| | 地址变换过程 | 访问一个逻辑地址的访存次数 |
|---|---|---|
| 基本地址变换机构 | ①算页号、页内偏移量<br>②检查页号合法性<br>③查页表，找到页面存放的内存块号<br>④根据内存块号与页内偏移量得到物理地址<br>⑤访问目标内存单元 | 两次访存 |
| 具有快表的地址变换机构 | ①算页号、页内偏移量<br>②检查页号合法性<br>③查快表。若命中，即可知道页面存放的内存块号，可直接进行⑤；若未命中则进行④<br>④查页表，找到页面存放的内存块号，并且将页表项复制到快表中<br>⑤根据内存块号与页内偏移量得到物理地址<br>⑥访问目标内存单元 | 快表命中，只需一次访存<br><br>快表未命中，需要两次访存 |

TLB 和 普通 Cache 的区别——TLB 中只有页表项的副本，而普通 Cache 中可能会有其他各种数据的副本

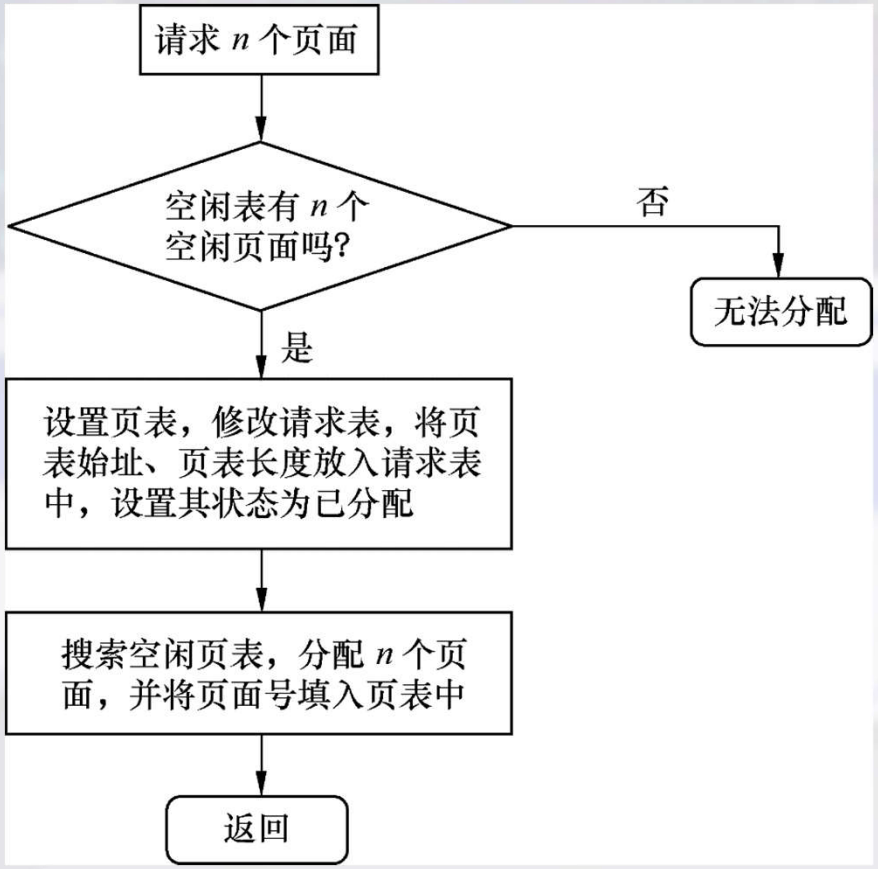
# 分页存储管理机制

## 本讲内容

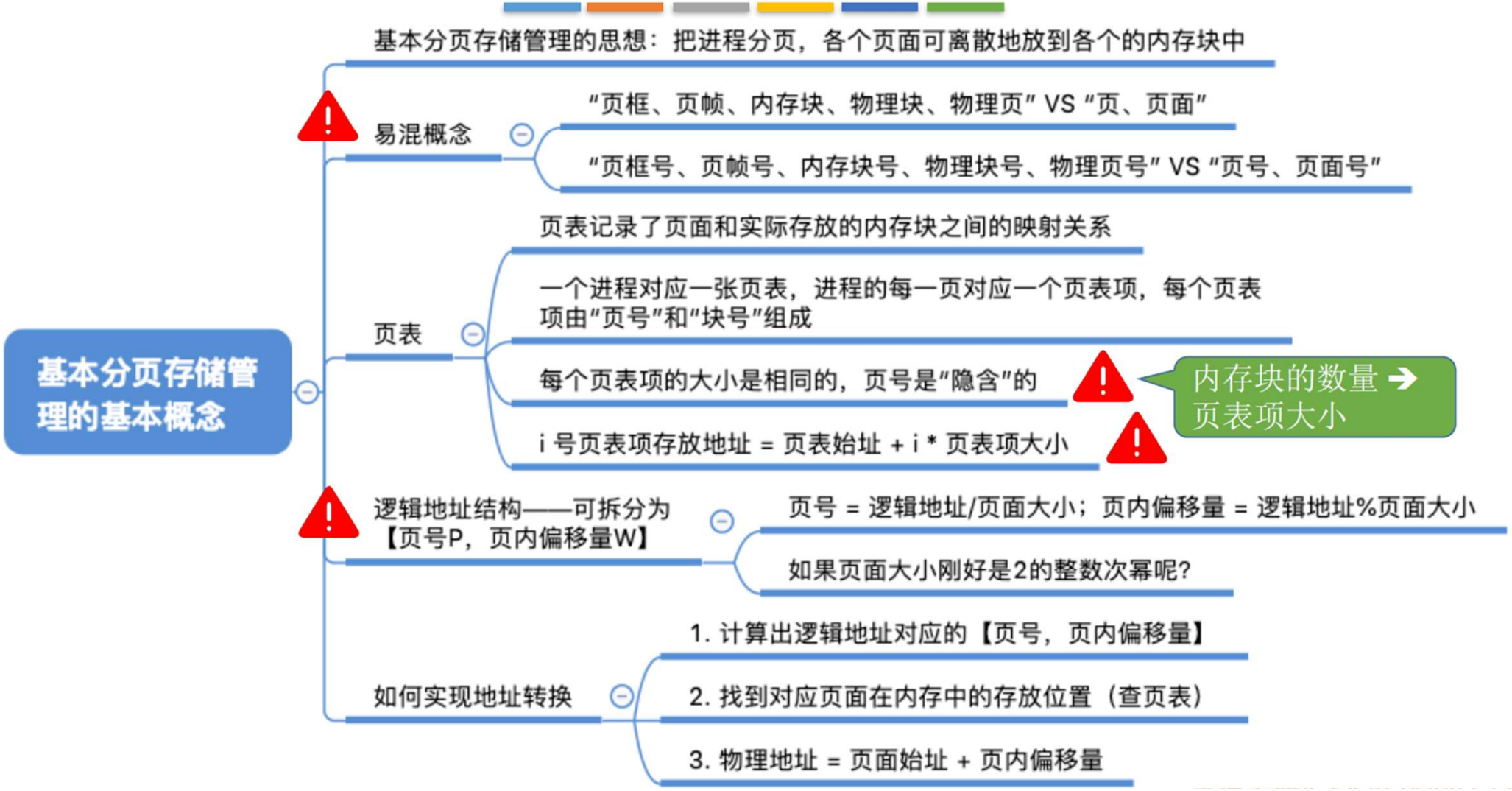
1. 逻辑页面与物理页框
2. 分页存储的管理表格
3. 分页存储的地址转换
4. 相联存储与快表技术
5. 物理页框的分配流程

# 物理页框的分配流程

- 计算所需要的页框数n
- 查位示图，是否有n个空闲页框
- 如果有足够的空闲页框，则页表长度设为n，填入PCB中
- 申请页表区，把页表始址填入请求表
- 分配n个空闲页框，将页框号和页号填入页表
- 修改位示图



# 知识回顾与重要考点


基本分页存储管理的基本概念
基本分页存储管理的思想：把进程分页，各个页面可离散地放到各个的内存块中
易混概念
“页框、页帧、内存块、物理块、物理页” VS “页、页面”
“页框号、页帧号、内存块号、物理块号、物理页号” VS “页号、页面号”
页表
页表记录了页面和实际存放的内存块之间的映射关系
一个进程对应一张页表，进程的每一页对应一个页表项，每个页表项由“页号”和“块号”组成
每个页表项的大小是相同的，页号是“隐含”的
内存块的数量 → 页表项大小
i 号页表项存放地址 = 页表始址 + i * 页表项大小
逻辑地址结构——可拆分为【页号P，页内偏移量W】
页号 = 逻辑地址/页面大小；页内偏移量 = 逻辑地址%页面大小
如果页面大小刚好是2的整数次幂呢?
如何实现地址转换
1. 计算出逻辑地址对应的【页号，页内偏移量】
2. 找到对应页面在内存中的存放位置（查页表）
3. 物理地址 = 页面始址 + 页内偏移量

8、某页式管理系统中，地址寄存器的低9位表示页内地址，则页面大小为（）。

A．1024字节

B．512字节

2^9=512字节

C．1024K字节

D．512K字节

6、设有8页的逻辑空间，每页有1024B，它们被映射到有32块的物理存储区中。那么，逻辑地址的有效位是（），物理地址至少是（ ）位。

A．10、11　　B．12、14

C．13、15　　D．14、16

逻辑地址：8*1024=2^3*2^10=2^13

物理地址：32*1024=2^5*2^10=2^15

7、一个分页存储管理系统中，地址长度为32位，其中页号占8位，则页表长度是（）。

A．2的8次方字节　　B．2的16次方字节

C．2的24次方字节　　D．2的32次方字节

页表长度 = 页表行数

# 计算页表大小

已知：

- 虚拟地址：32-bit
- 物理地址：30-bit
- 页面大小：4 KB
- 页表中每个记录占：4B

求：每个进程的页表需占用多大存储空间？

# 计算页表大小

1.计算页面数:

1.虚拟地址空间为32位，因此每个进程可访问的总虚拟地址大小为2^32 字节。

2.页面大小为4 KB，即2^12 字节。

3.总页面数 = 2^32 / 2^12 = 2^20 个页面

2.计算页表占用的总存储空间:

1.页表中每个记录占4字节，即2^2 字节。

2.每个进程的页表存储空间大小 = 总页面数 * 每个记录大小

3.每个进程的页表存储空间大小 = 2^20 * 2^2 = 2^22 字节

因此，每个进程的页表需占用2^22 字节的存储空间。