第三章

存储管理

方　钰

主 要 内 容

3.1 存储管理的主要任务

3.2 连续分配方式

3.3 页式存储管理

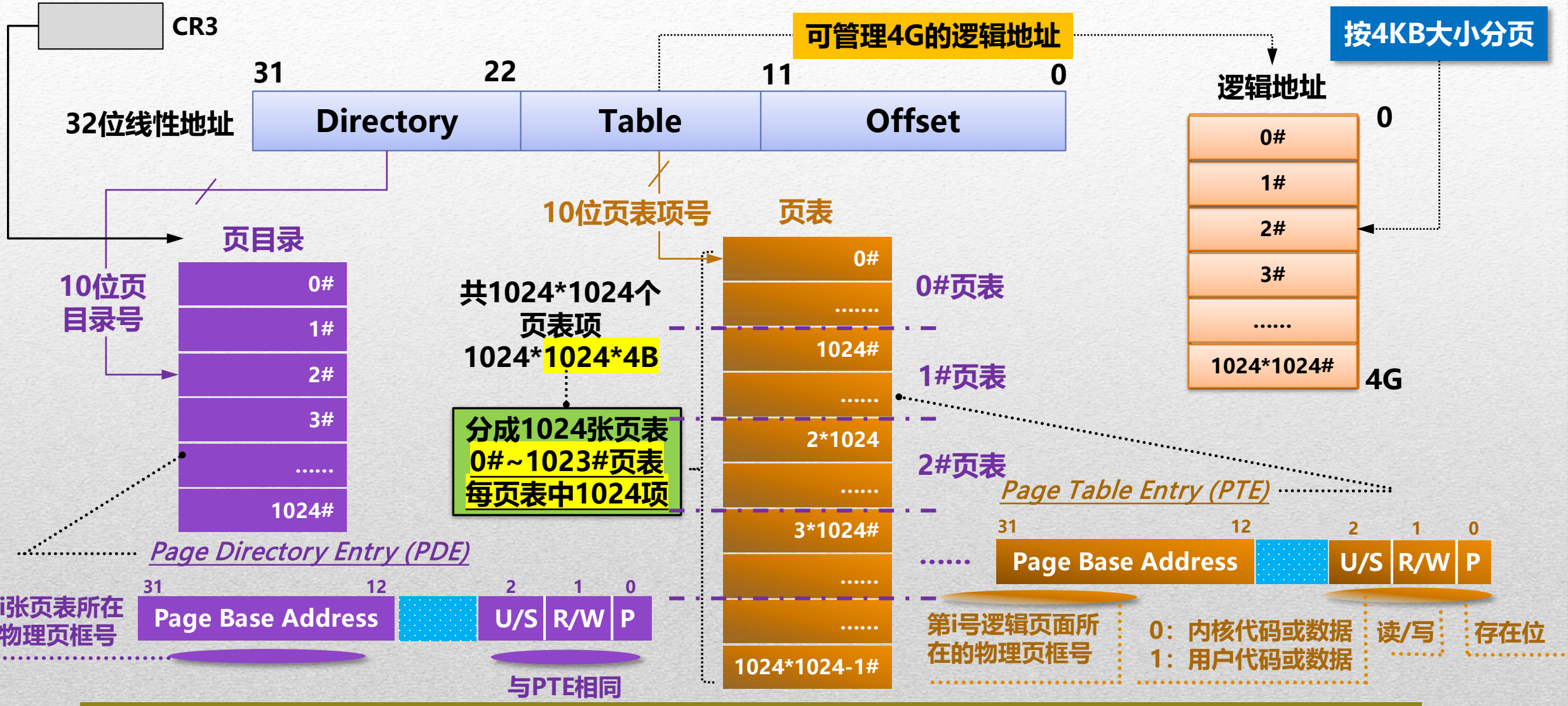
3.4 段式与段页式存储管理**

3.5 UNIX 存储管理

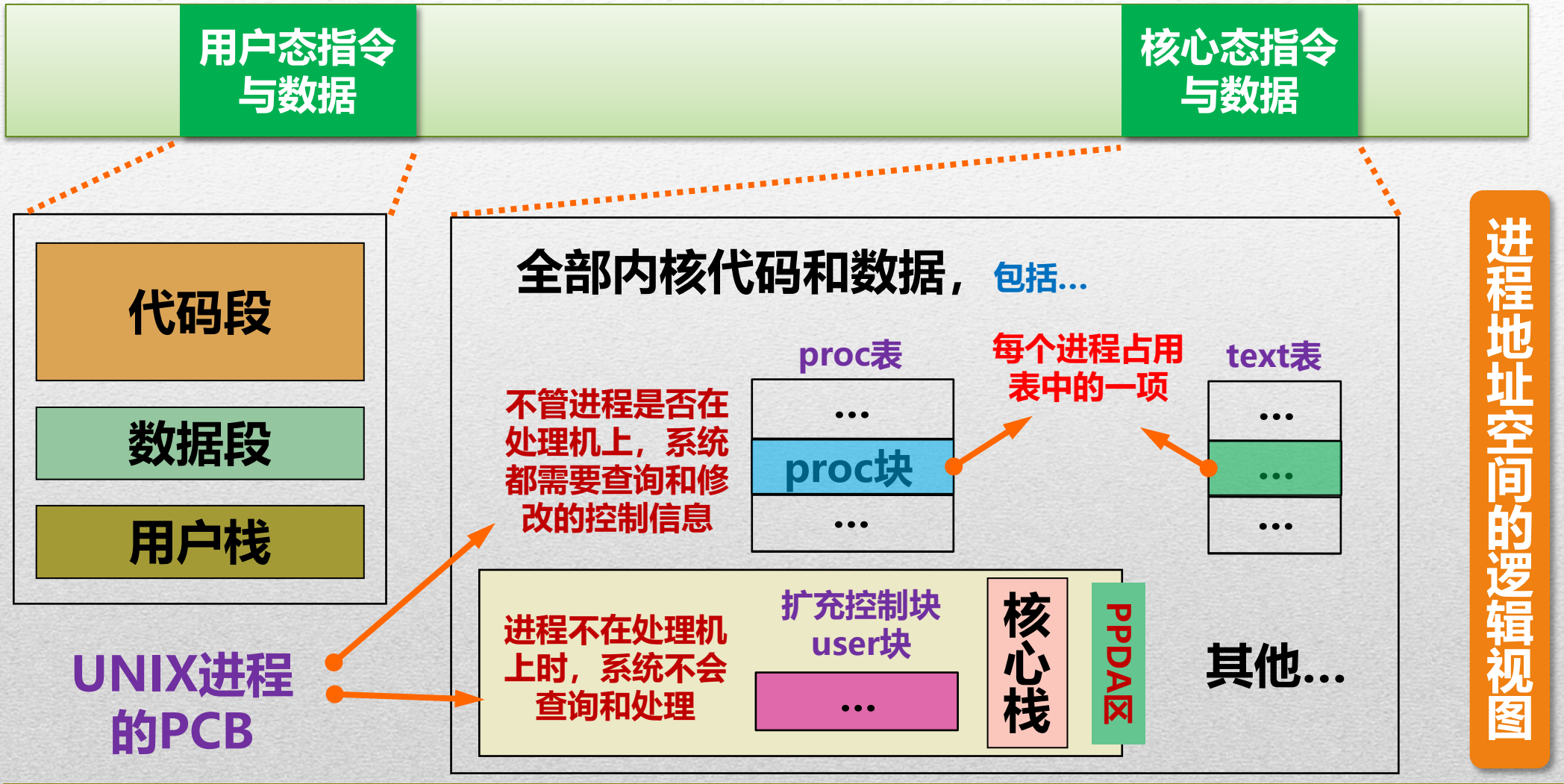
- 程序地址空间
- 物理地址空间
- 地址变换
- 存储空间管理

i386线性地址空间

使用二级页表管理进程的线性地址

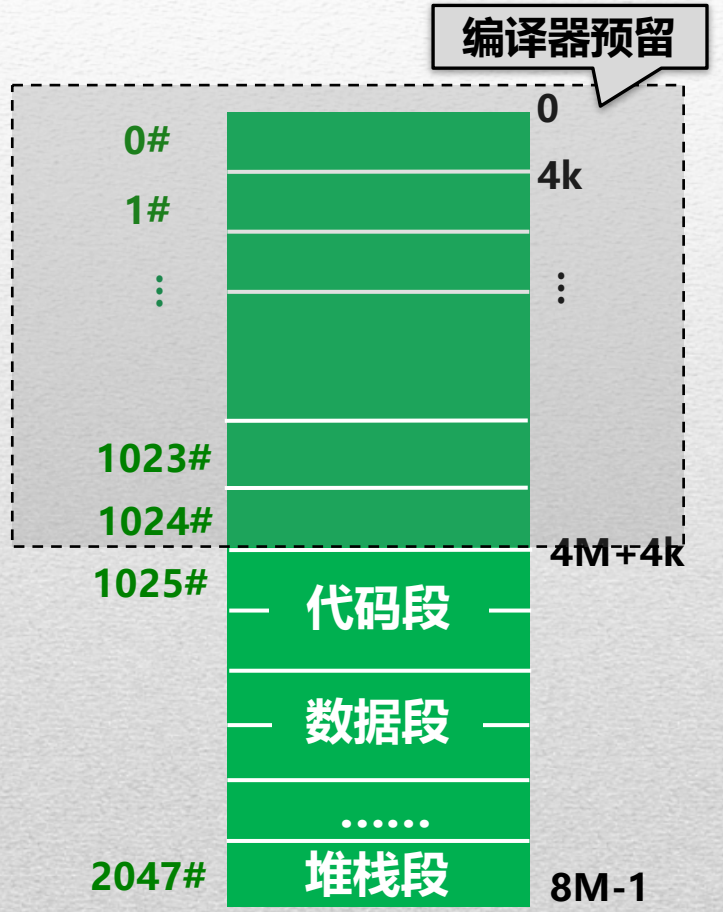


复习一下UNIX V6++的进程程序地址空间……

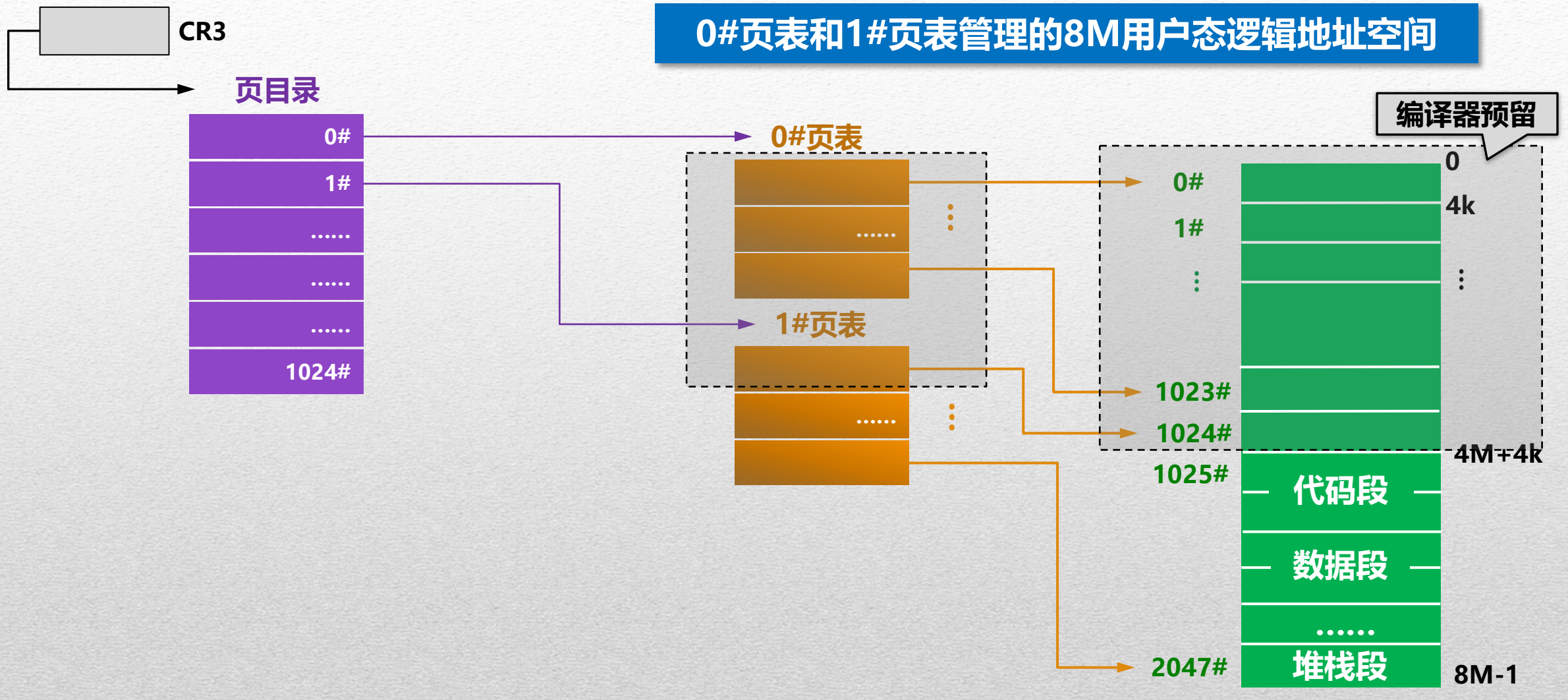


UNIX V6++的程序地址空间（线性地址、逻辑地址）

8M用户态逻辑地址空间

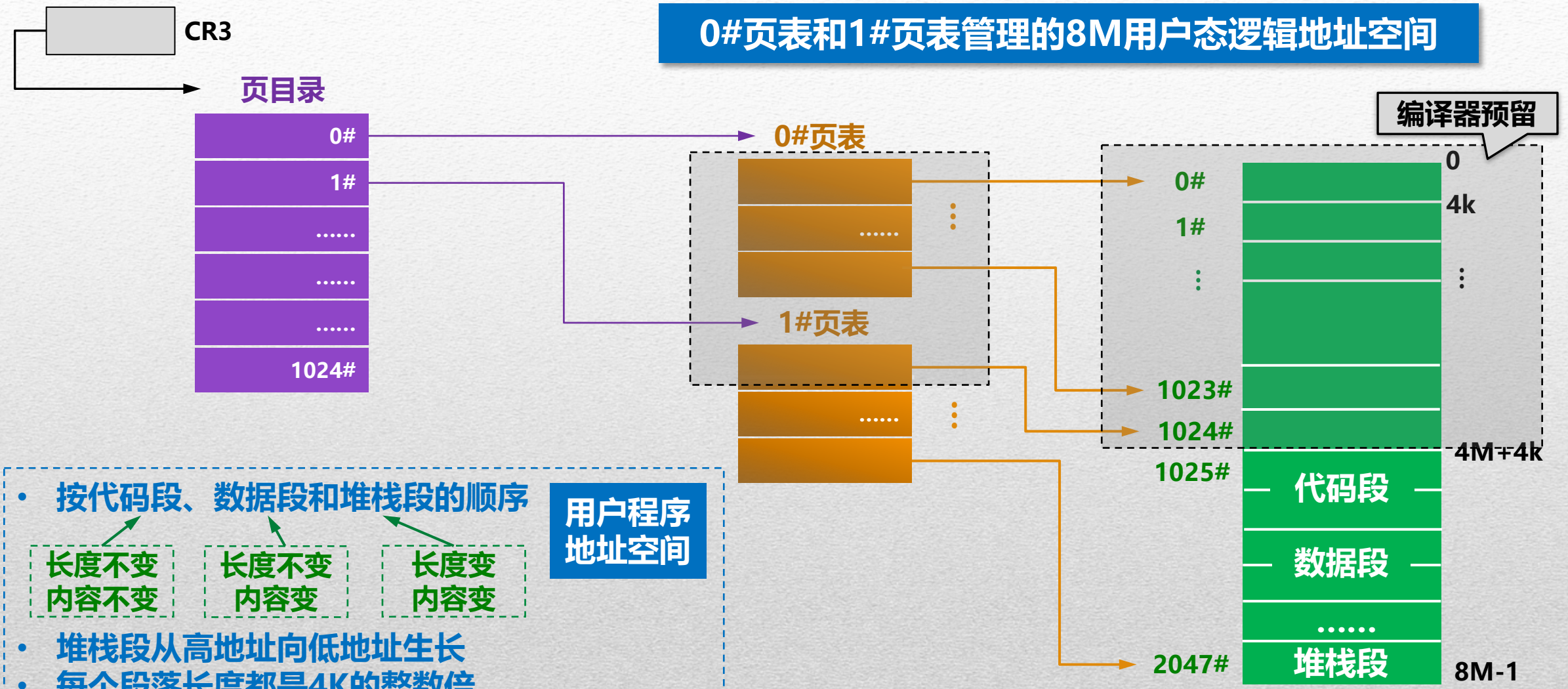


UNIX V6++的程序地址空间（线性地址、逻辑地址）

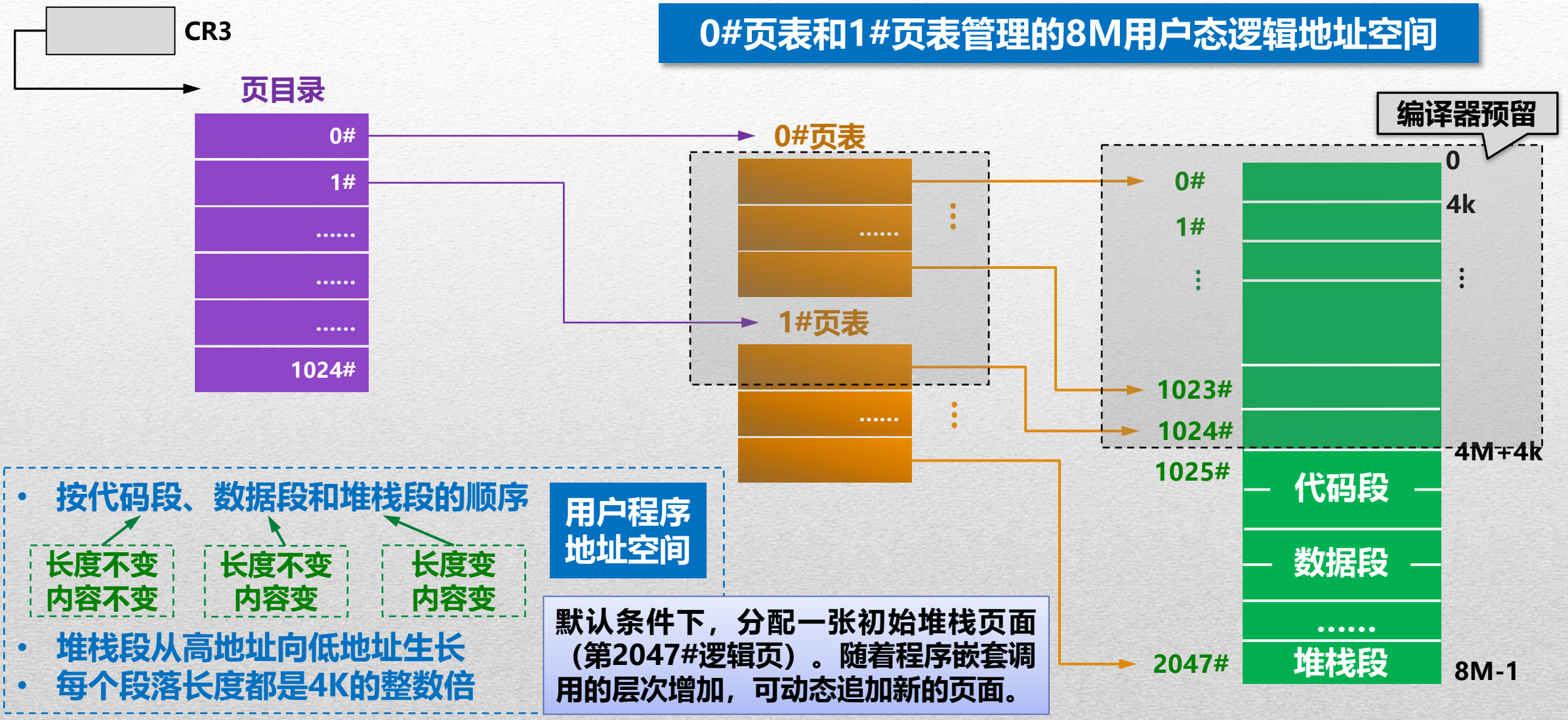


0#页表和1#页表管理的8M用户态逻辑地址空间

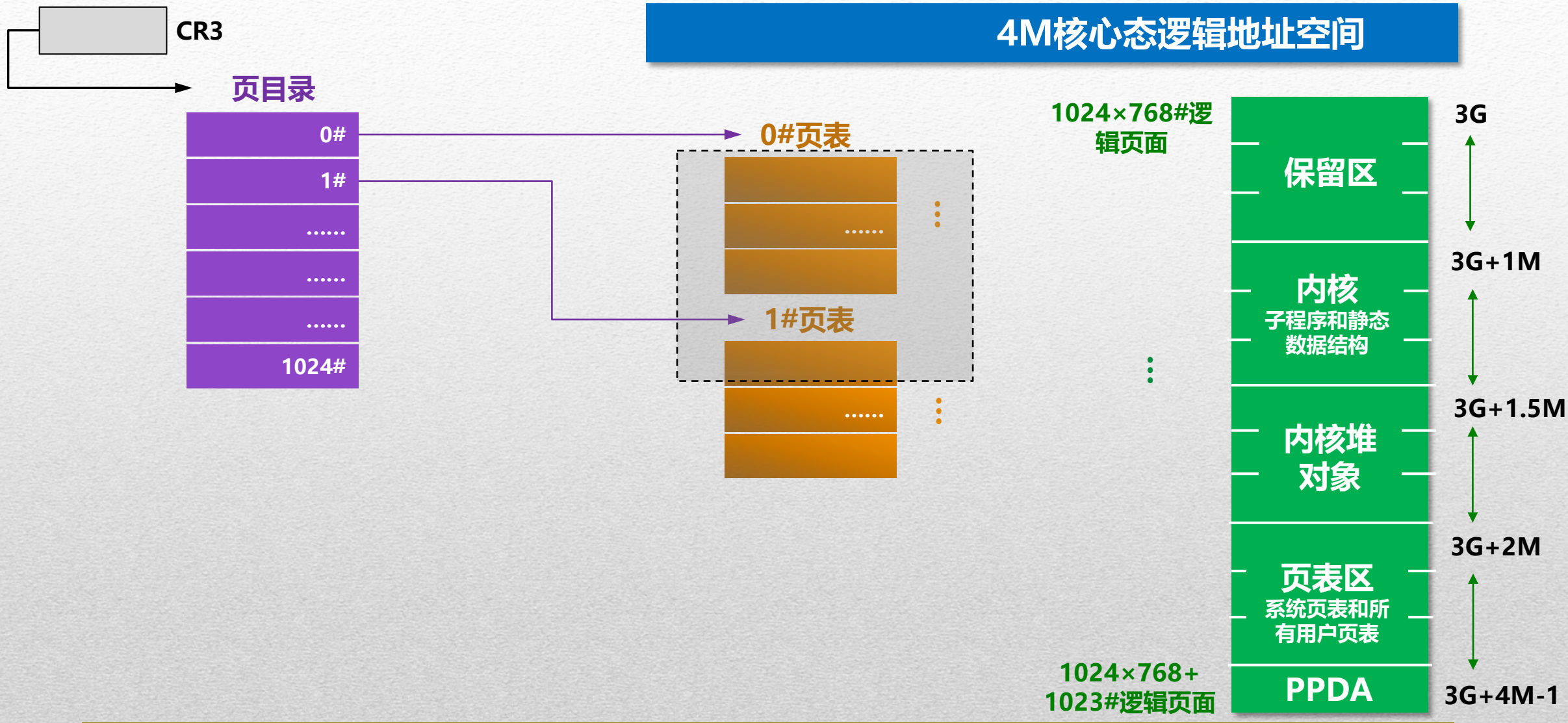
UNIX V6++的程序地址空间（线性地址、逻辑地址）



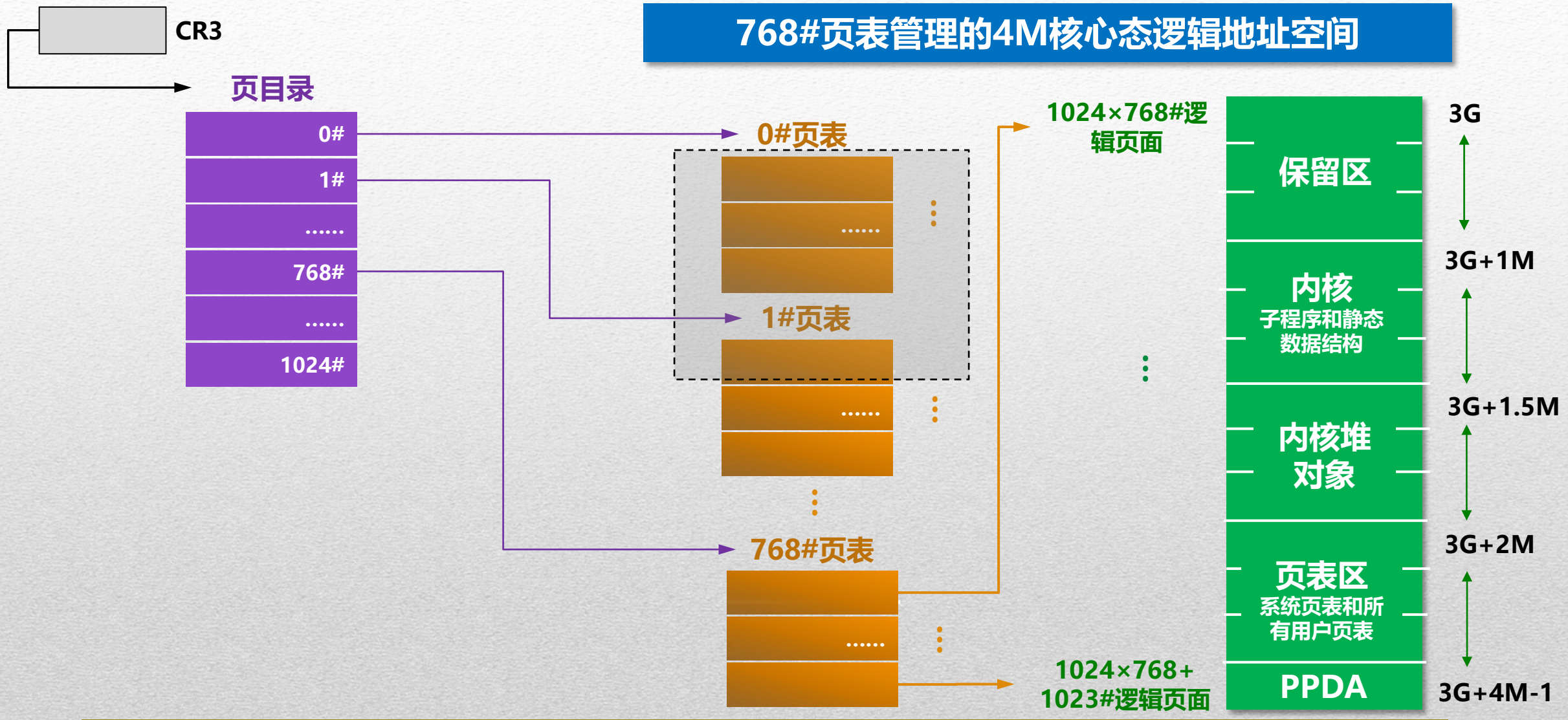
UNIX V6++的程序地址空间（线性地址、逻辑地址）



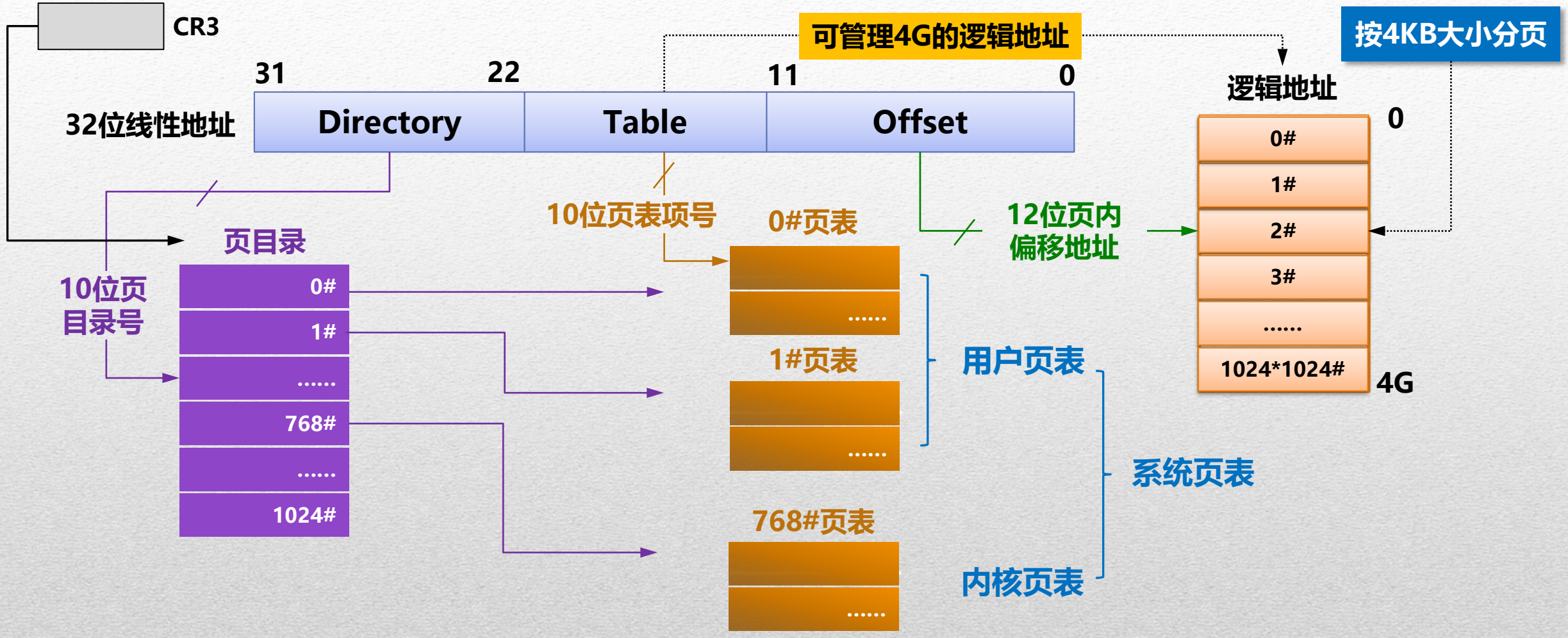
UNIX V6++的程序地址空间（线性地址、逻辑地址）



UNIX V6++的程序地址空间（线性地址、逻辑地址）



UNIX V6++的程序地址空间（线性地址、逻辑地址）



主 要 内 容

3.1 存储管理的主要任务

3.2 连续分配方式

3.3 页式存储管理

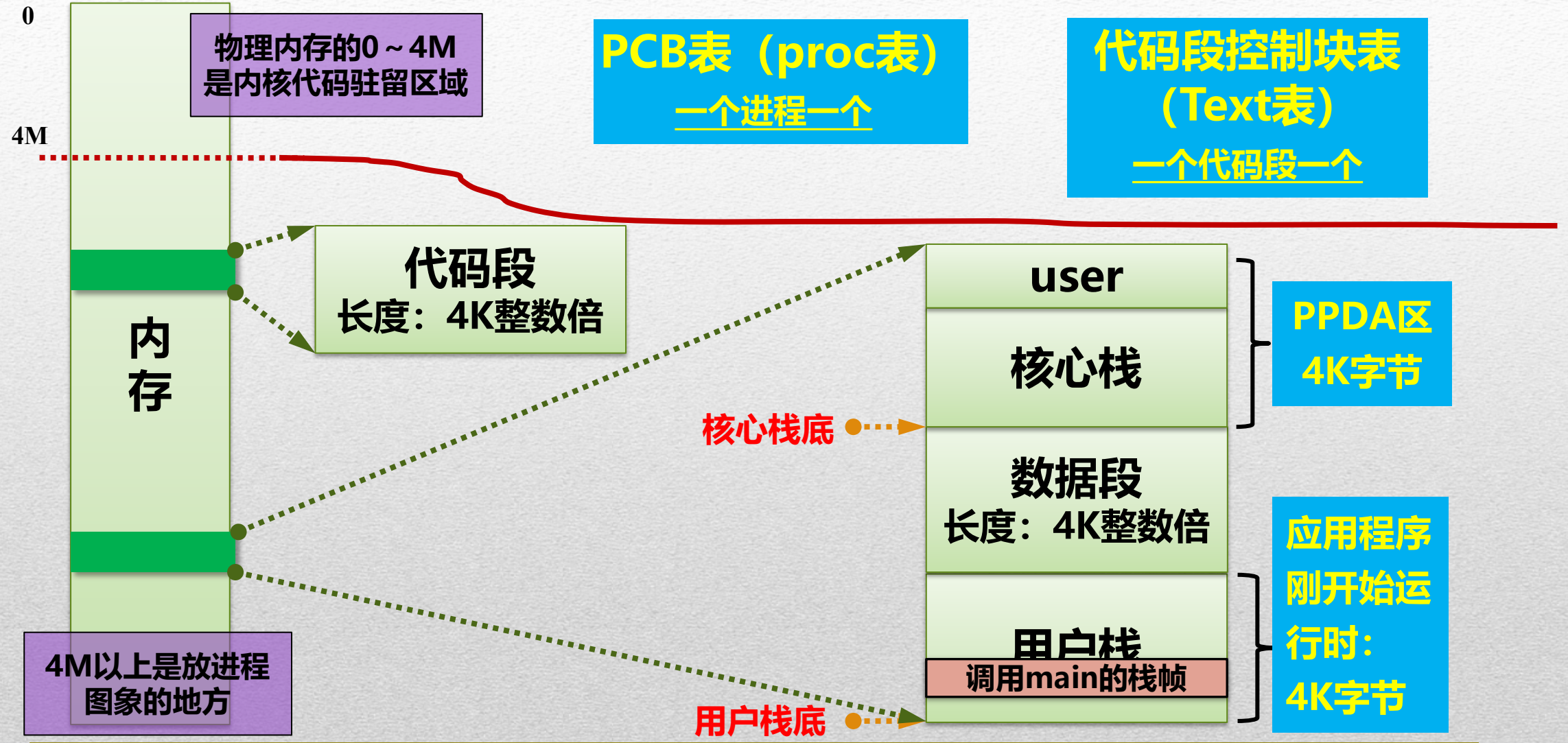
3.4 段式与段页式存储管理**

3.5 UNIX 存储管理

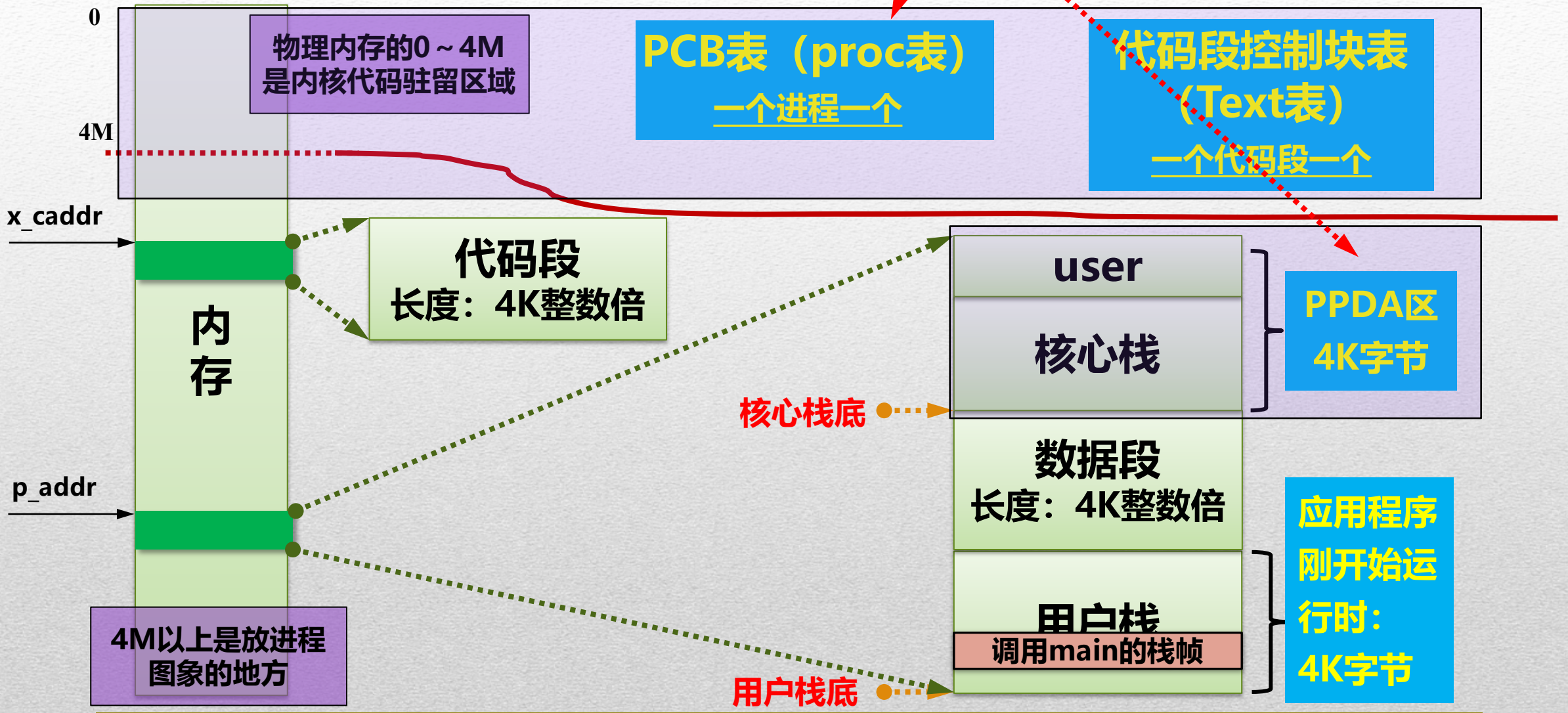
- 程序地址空间
- 物理地址空间
- 地址变换
- 存储空间管理

UNIX中进程的构成（进程图象）

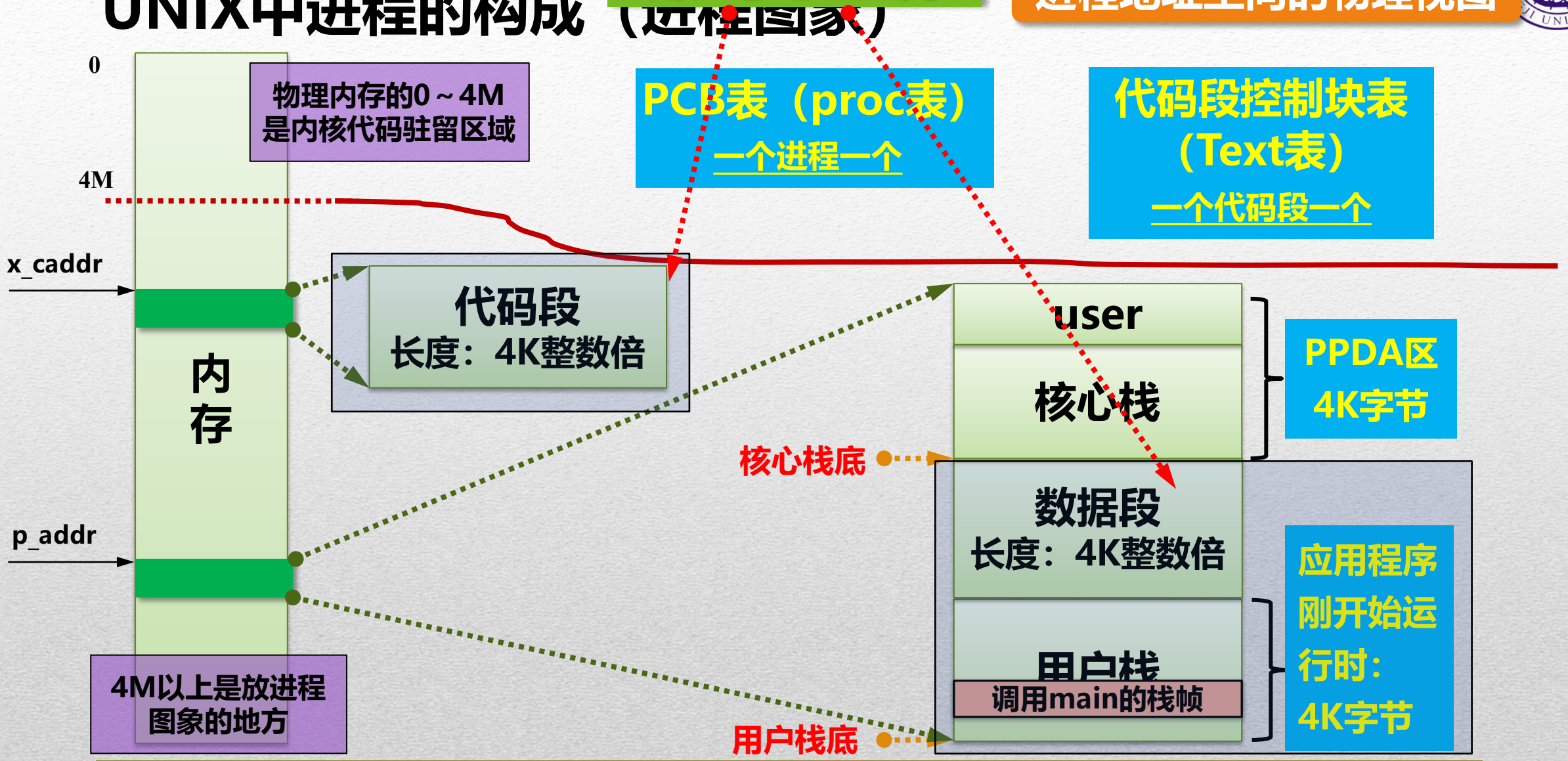
进程地址空间的物理视图



UNIX中进程的构成（进程图象）



UNIX中进程的构成（进程图象）



主 要 内 容

3.1 存储管理的主要任务

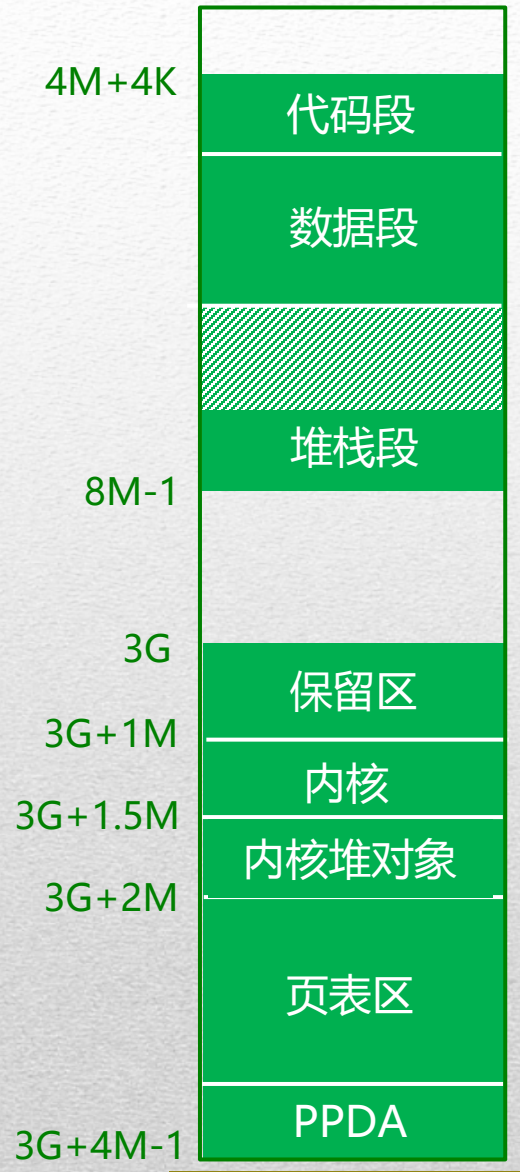
3.2 连续分配方式

3.3 页式存储管理

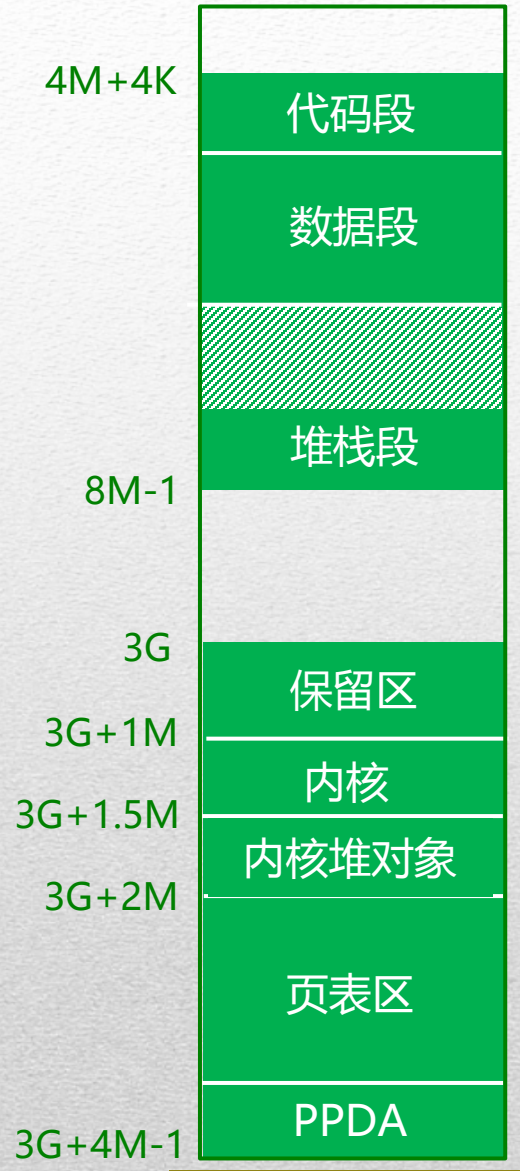
3.4 段式与段页式存储管理**

3.5 UNIX 存储管理

- 程序地址空间
- 物理地址空间
- 地址变换
- 存储空间管理


4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
页表区
PPDA
3G+4M-1


0
保留区
1M
内核
1.5M
内核堆对象
2M
页表区
4M
代码段
x_caddr
用户区
PPDA
p_addr
数据段
堆栈段

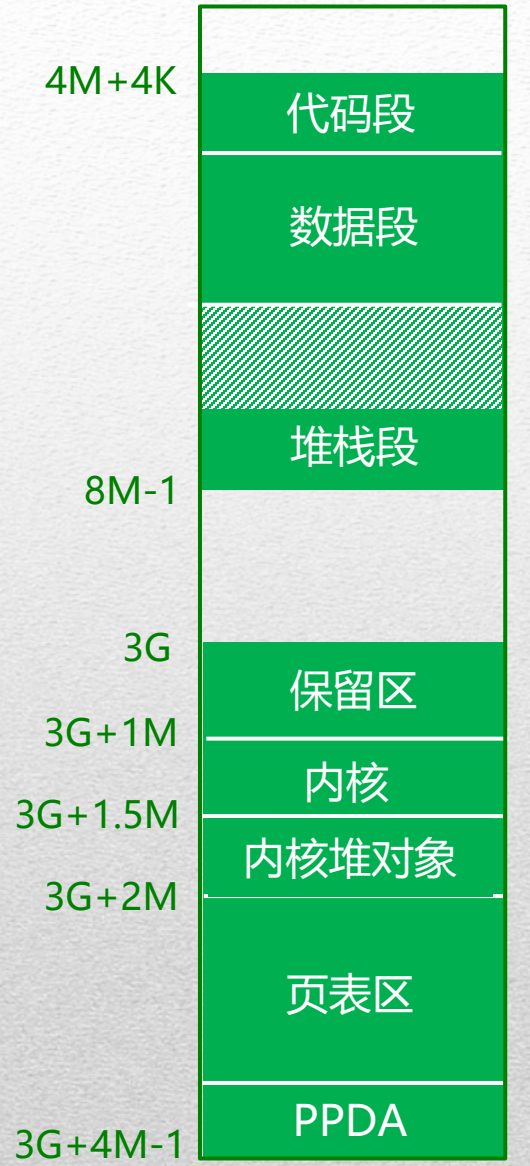


Page Table 0#

	Page Base Address		u/s	r/w	p
0#					
1#					
1023#					

Page Table 1#

	Page Base Address		u/s	r/w	p
0#					
1#					
1023#					



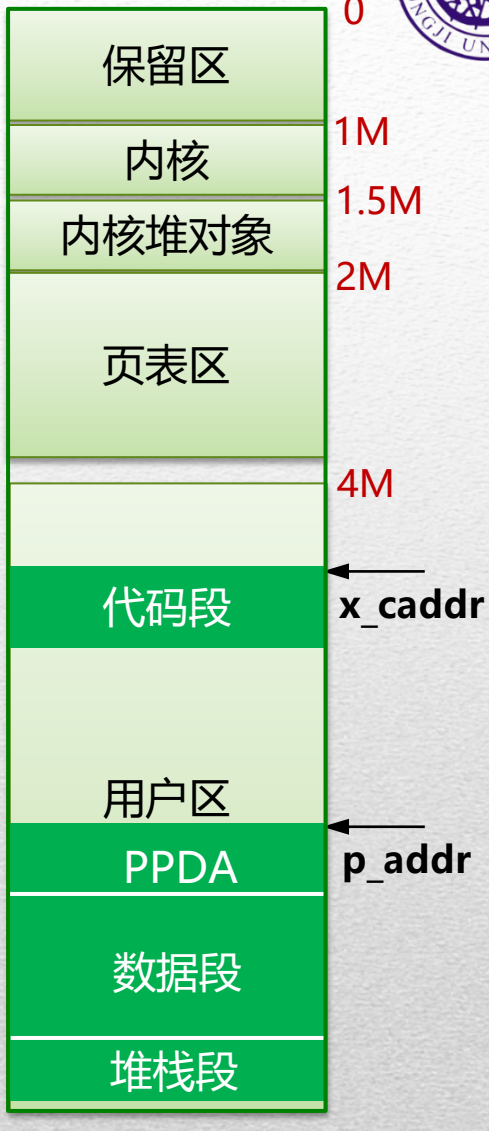
Page Table 0#

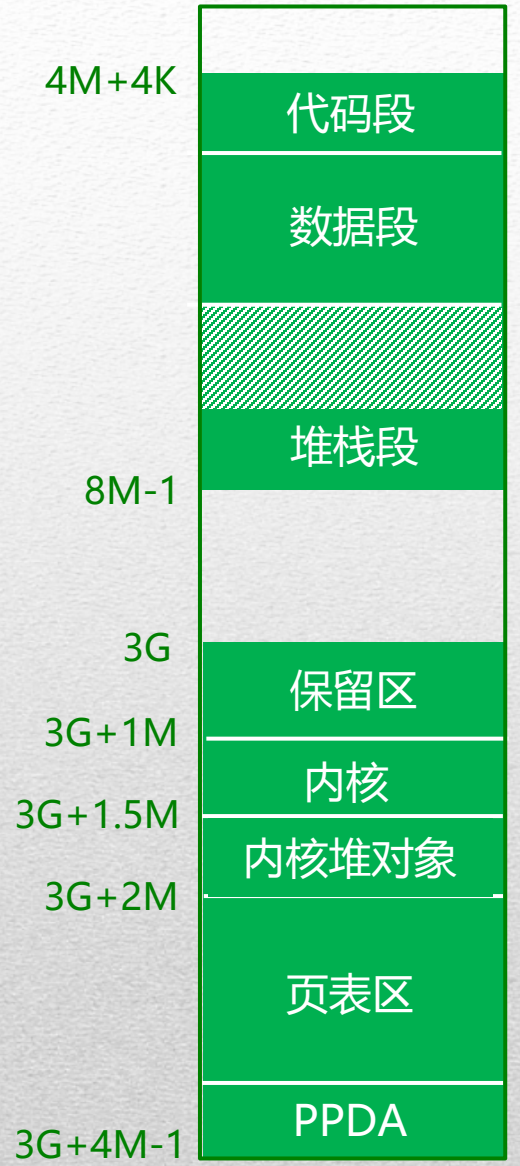
	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

编译器预留

Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#					
1023#					





Page Table 0#

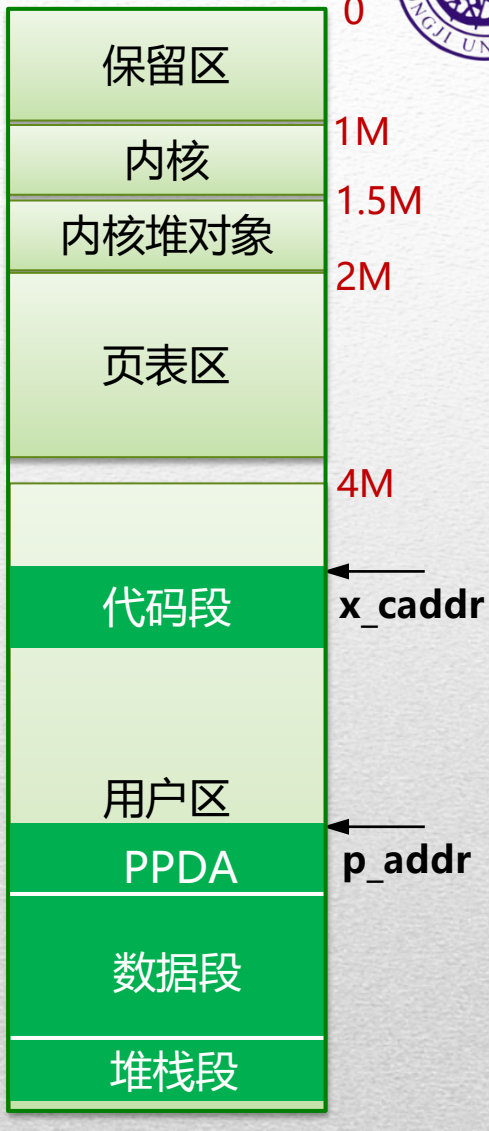
	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

编译器预留

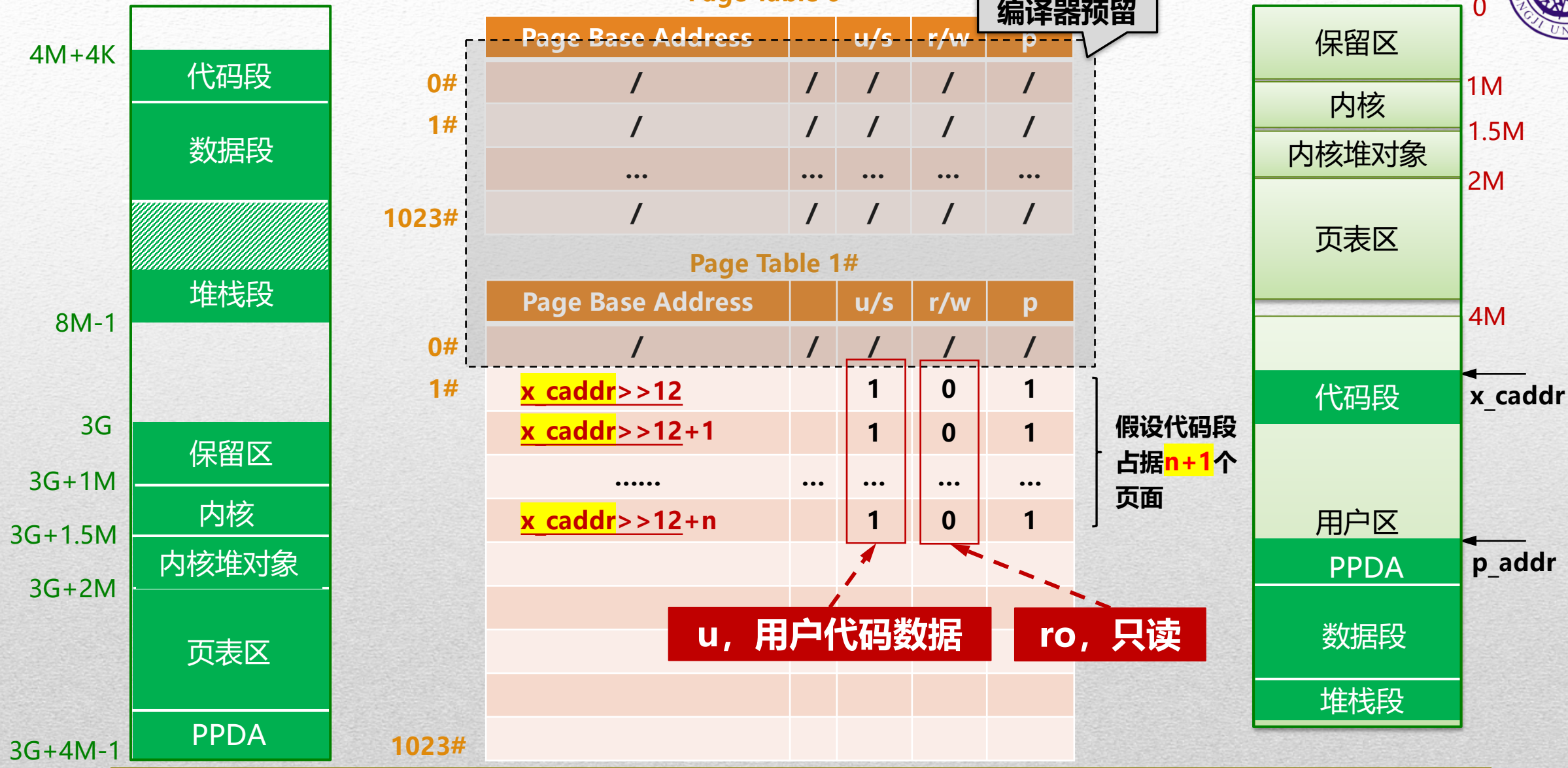
Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#					
1023#					

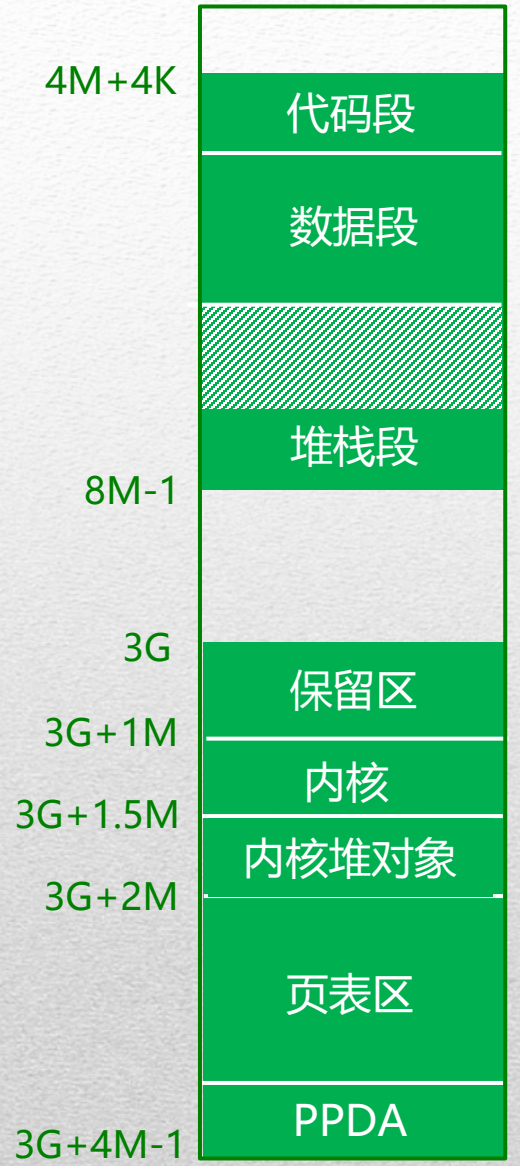
假设代码段占据n+1个页面




4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
页表区
PPDA
3G+4M-1
Page Table 0#
编译器预留
Page Base Address
u/s
r/w
p
0#
1#
1023#
Page Table 1#
假设代码段占据n+1个页面
u，用户代码数据
ro，只读
0
保留区
1M
内核
1.5M
内核堆对象
2M
页表区
4M
代码段
x_caddr
用户区
PPDA
p_addr
数据段
堆栈段


Page Table 0#
编译器预留
Page Base Address
u/s
r/w
p
0#
1#
1023#
Page Table 1#
x_caddr>>12
x_caddr>>12+1
x_caddr>>12+n
假设代码段占据n+1个页面
u，用户代码数据
ro，只读
4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
页表区
PPDA
3G+4M-1
0
1M
1.5M
2M
4M
x_caddr
用户区
p_addr


Page Table 0#
编译器预留
Page Base Address | | u/s | r/w | p
0# / / / / /
1# / / / / /
... ...
1023# / / / / /
Page Table 1#
Page Base Address | | u/s | r/w | p
0# / / / / /
1# x_caddr>>12 1 0 1
x_caddr>>12+1 1 0 1
....
x_caddr>>12+n 1 0 1
1023#
从x_caddr>>12开始的n+1个连续的物理页框
假设代码段占据n+1个页面
u，用户代码数据
ro，只读
4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
页表区
PPDA
3G+4M-1
0
保留区
1M
内核
1.5M
内核堆对象
2M
页表区
4M
代码段
x_caddr
用户区
PPDA
p_addr
数据段
堆栈段



Page Table 0#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

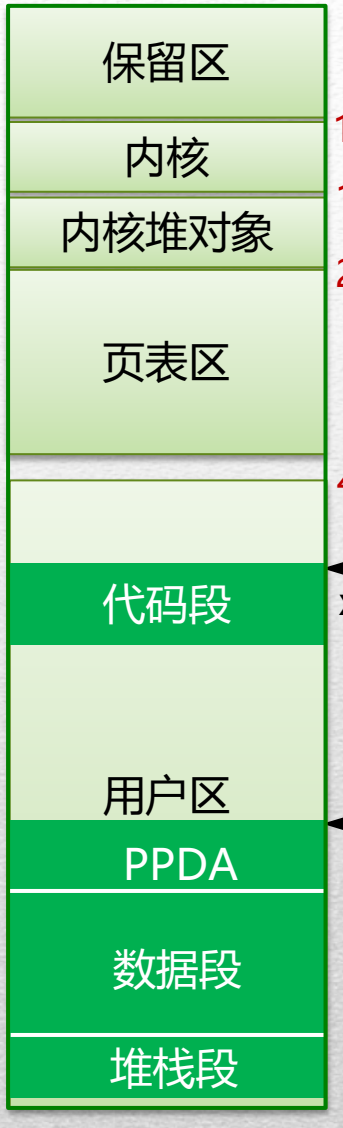
编译器预留

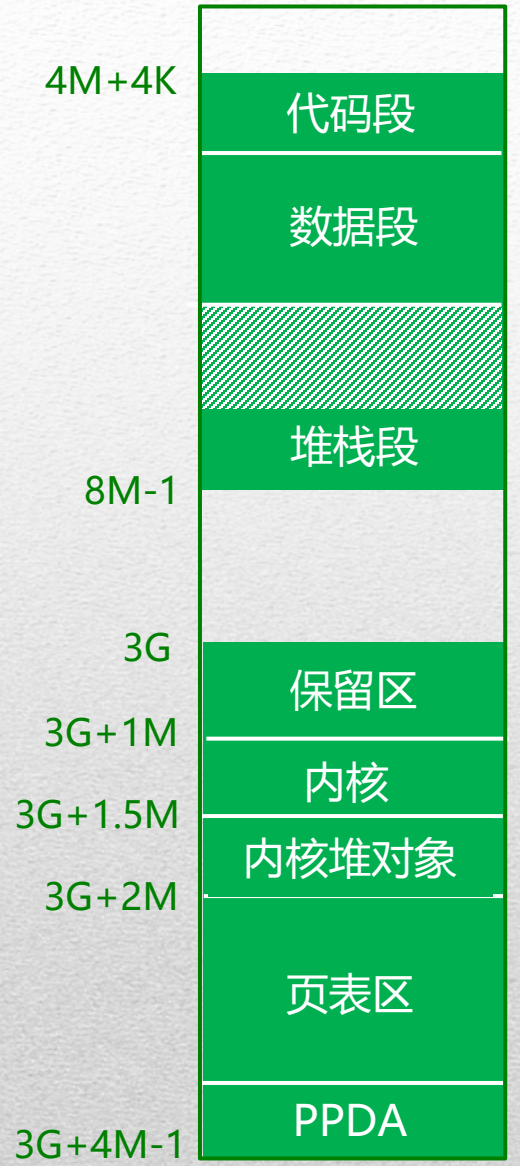
Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
1023#					

假设代码段占据n+1个页面

假设数据段占据m+1个页面





Page Table 0#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

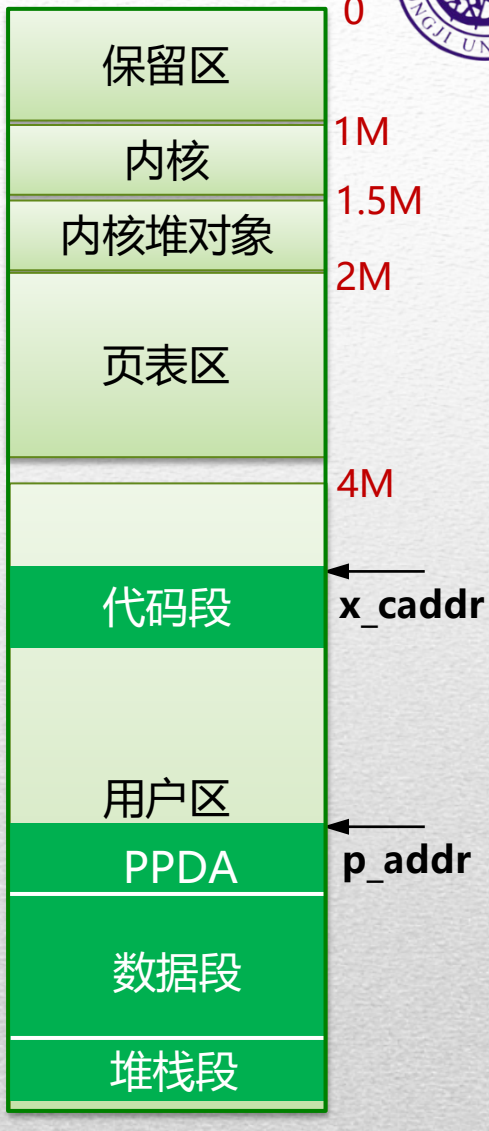
编译器预留

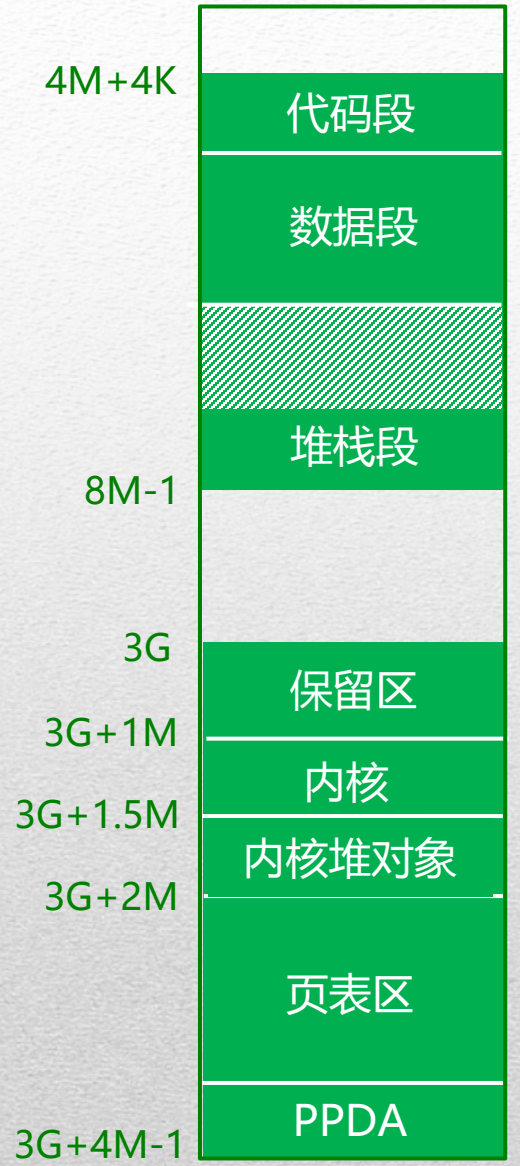
Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
			1	1	1
	.....	...	...	...	...
	.....	...	...	...	...
1023#					

假设代码段占据n+1个页面

假设数据段占据m+1个页面





Page Table 0#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

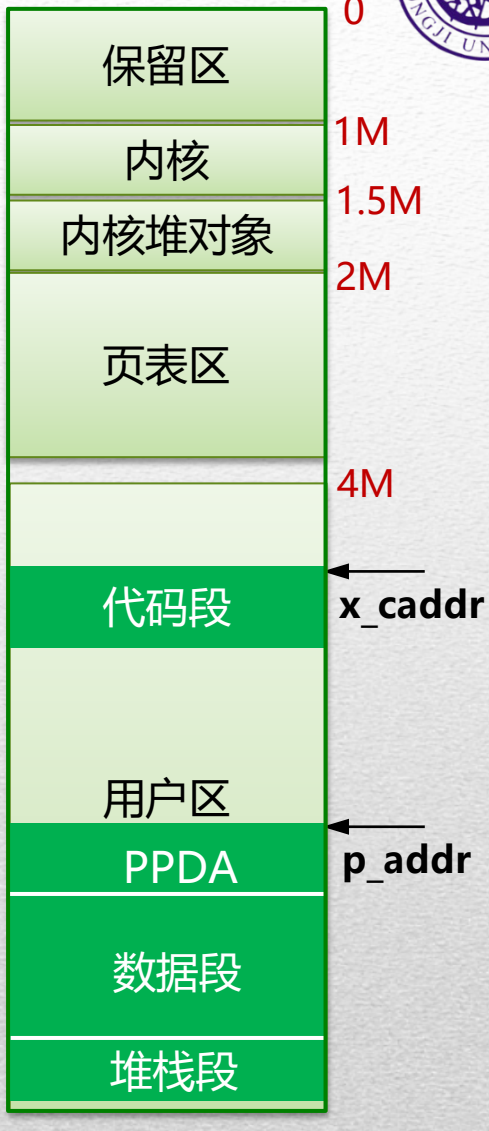
编译器预留

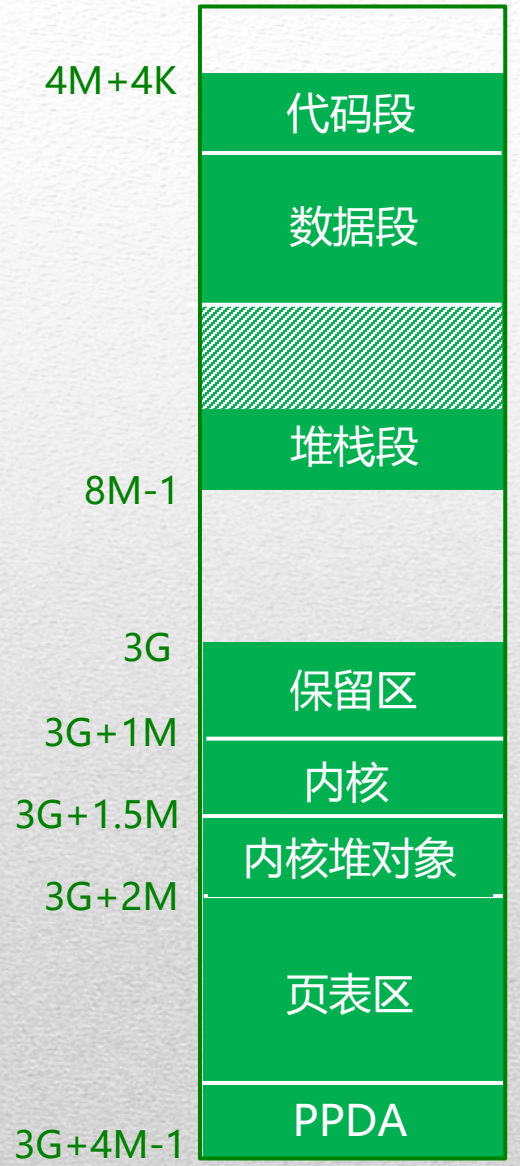
Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
	p_addr >>12+1		1	1	1
	.....	...	...	...	...
	p_addr >>12+1+m				
	.....	...	...	...	...
1023#					

假设代码段占据n+1个页面

假设数据段占据m+1个页面





Page Table 0#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

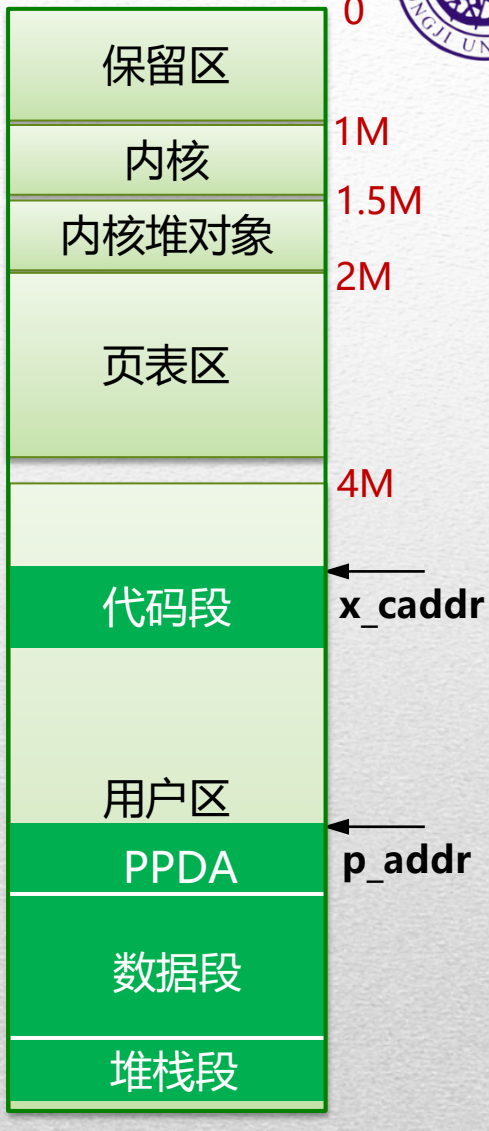
编译器预留

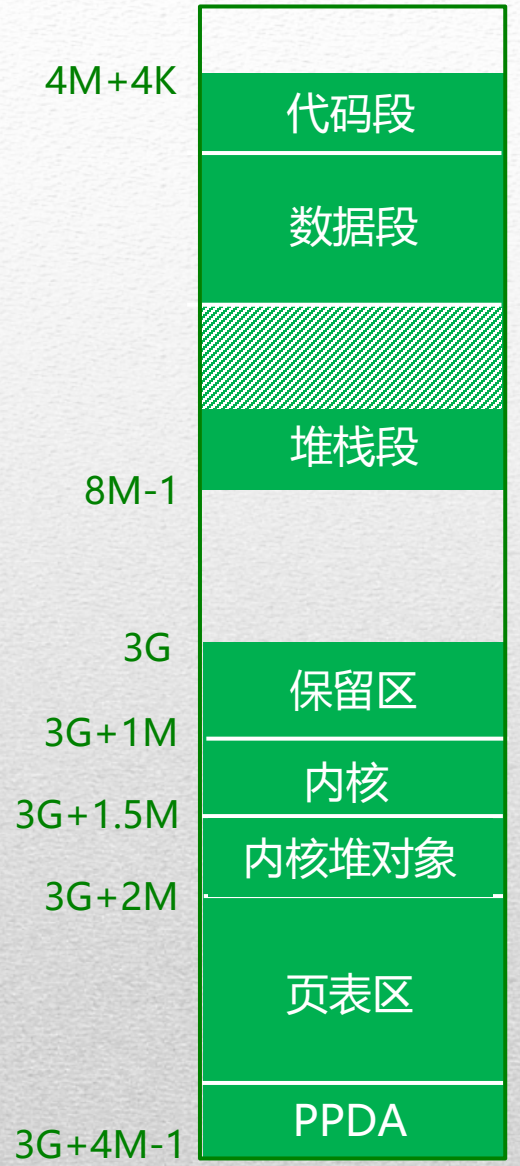
Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
	p_addr >>12+1		1	1	1
	.....	...	...	...	...
	p_addr >>12+1+m				
	.....	...	...	...	...
1023#	p_addr >>12+1+m+1		1	1	1

假设代码段占据n+1个页面

假设数据段占据m+1个页面





Page Table 0#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	/	/	/	/	/
	...	...	...	...	...
1023#	/	/	/	/	/

编译器预留

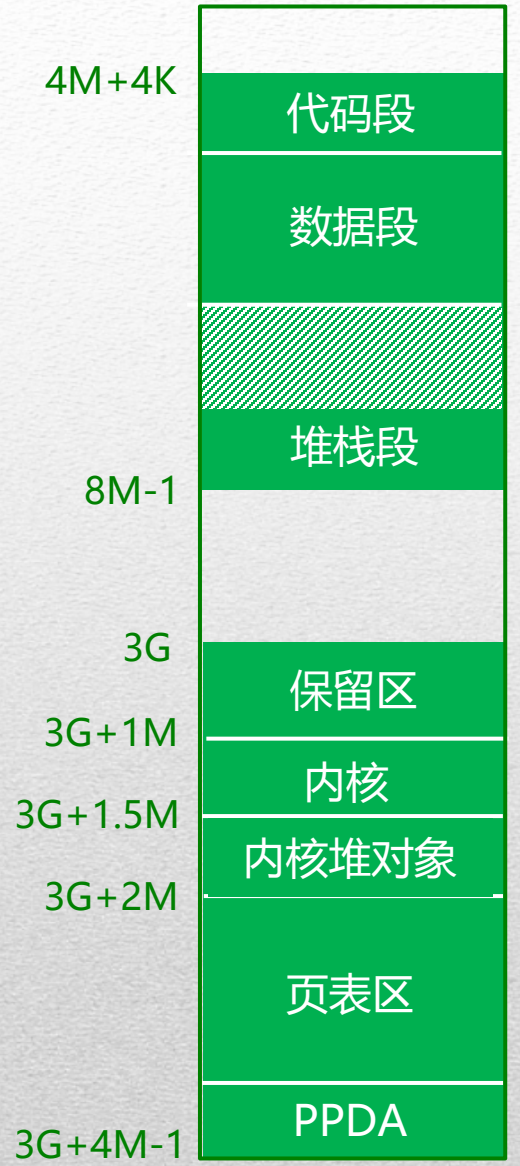
Page Table 1#

	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
	p_addr >>12+1		1	1	1
	.....	...	...	...	...
	p_addr >>12+1+m				
	.....	...	...	...	...
1023#	p_addr >>12+1+m+1		1	1	1

假设代码段占据n+1个页面

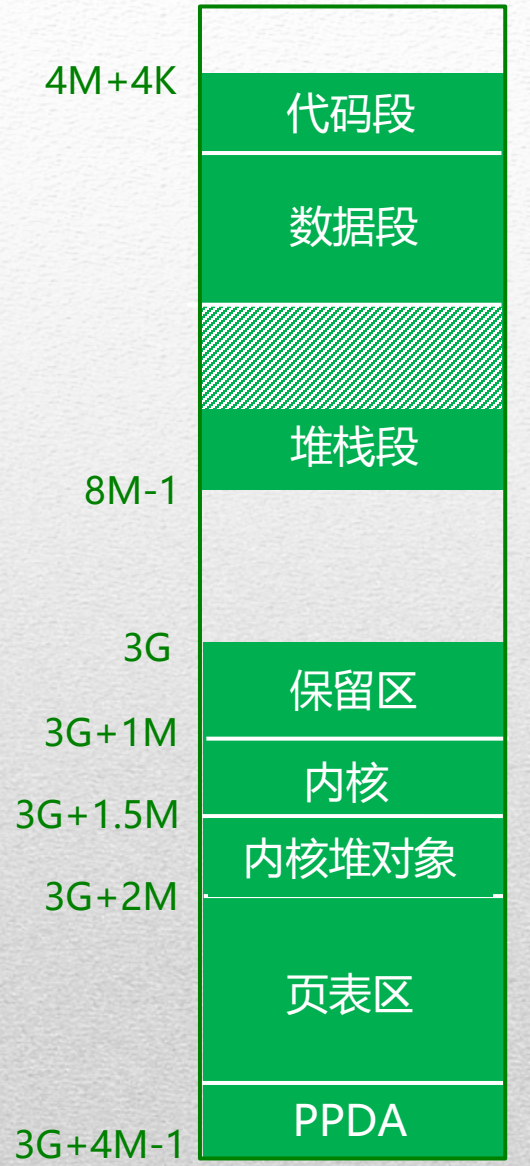
假设数据段占据m+1个页面

从p_addr>>12+1开始的m+2个连续的物理页框



Page Table 768#

	Page Base Address		u/s	r/w	p
0#					
1#					
1022#					
1023#					

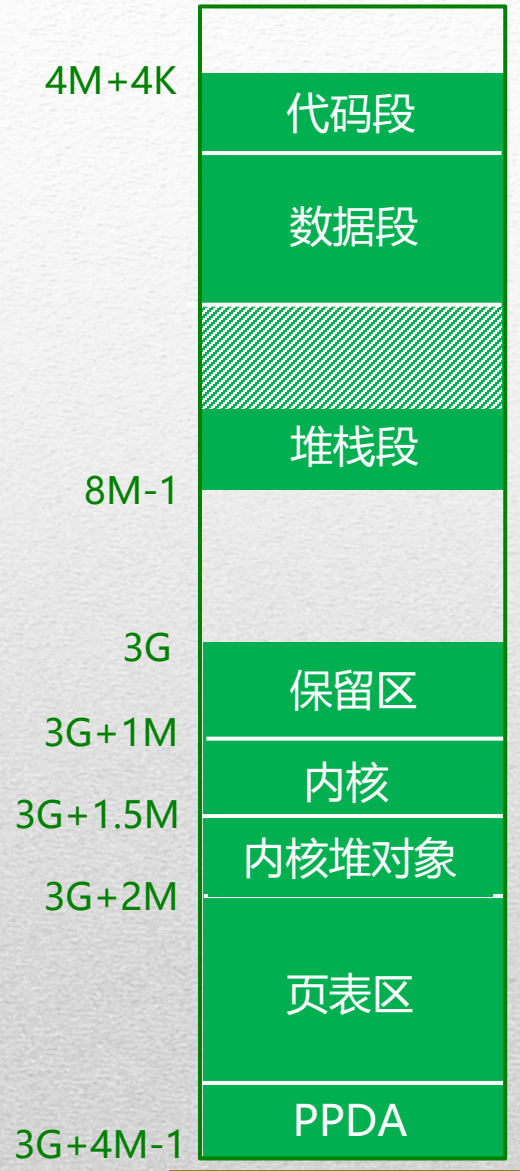


Page Table 768#

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	2		0	1	1
	.....	...	...	...	...
1022#					
1023#					

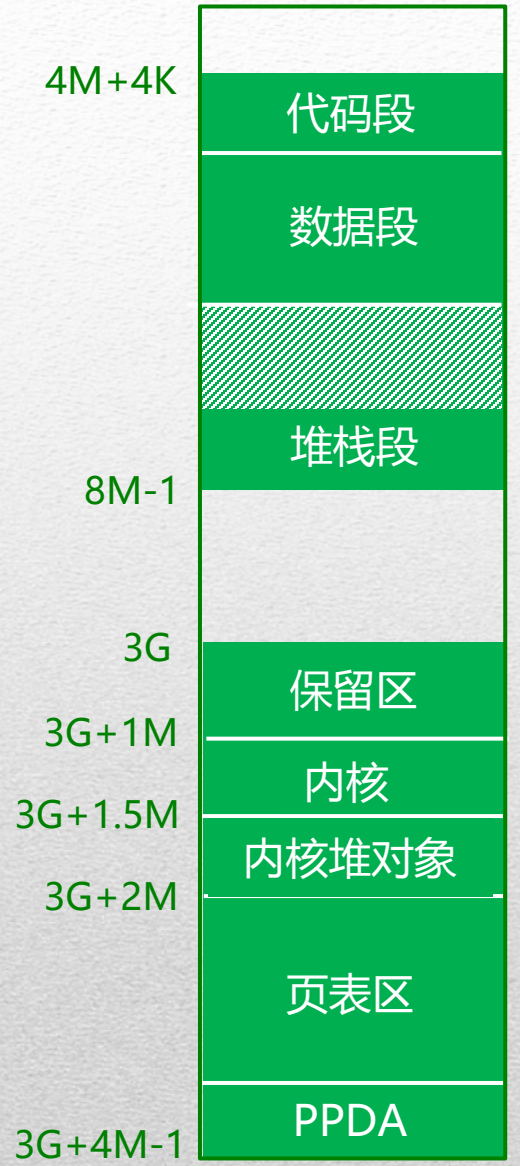
s，内核代码数据

rw，读写



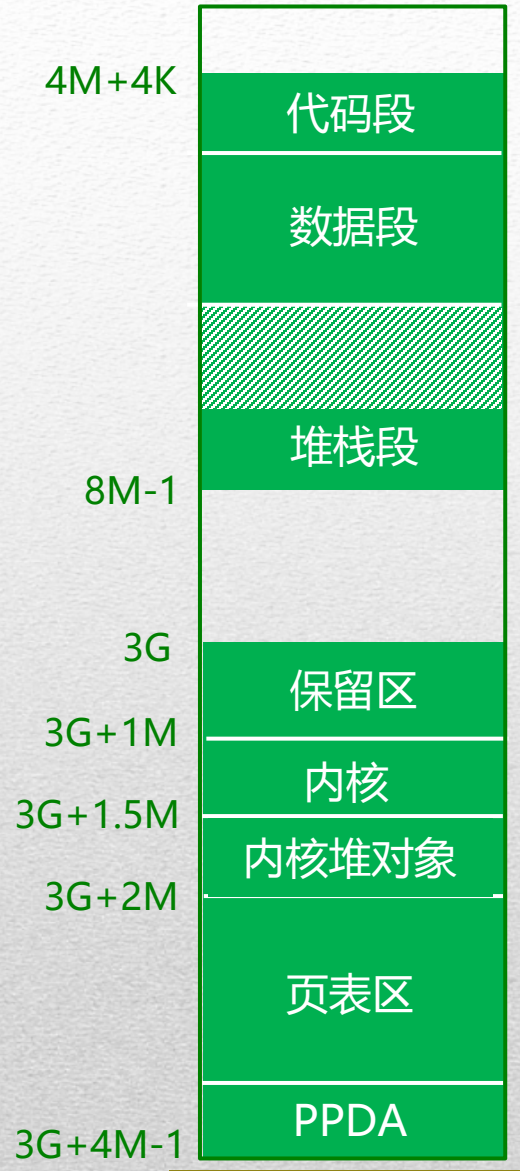
Page Table 768#

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	2		0	1	1
	.....	...	...	...	...
1022#	1022		0	1	1
1023#					



Page Table 768#

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	2		0	1	1
	.....	...	...	...	...
1022#	1022		0	1	1
1023#	p_addr>>12		0	1	1



Page Table 0#

Page Base Address		u/s	r/w	p
/	/	/	/	/

Page Table 1#

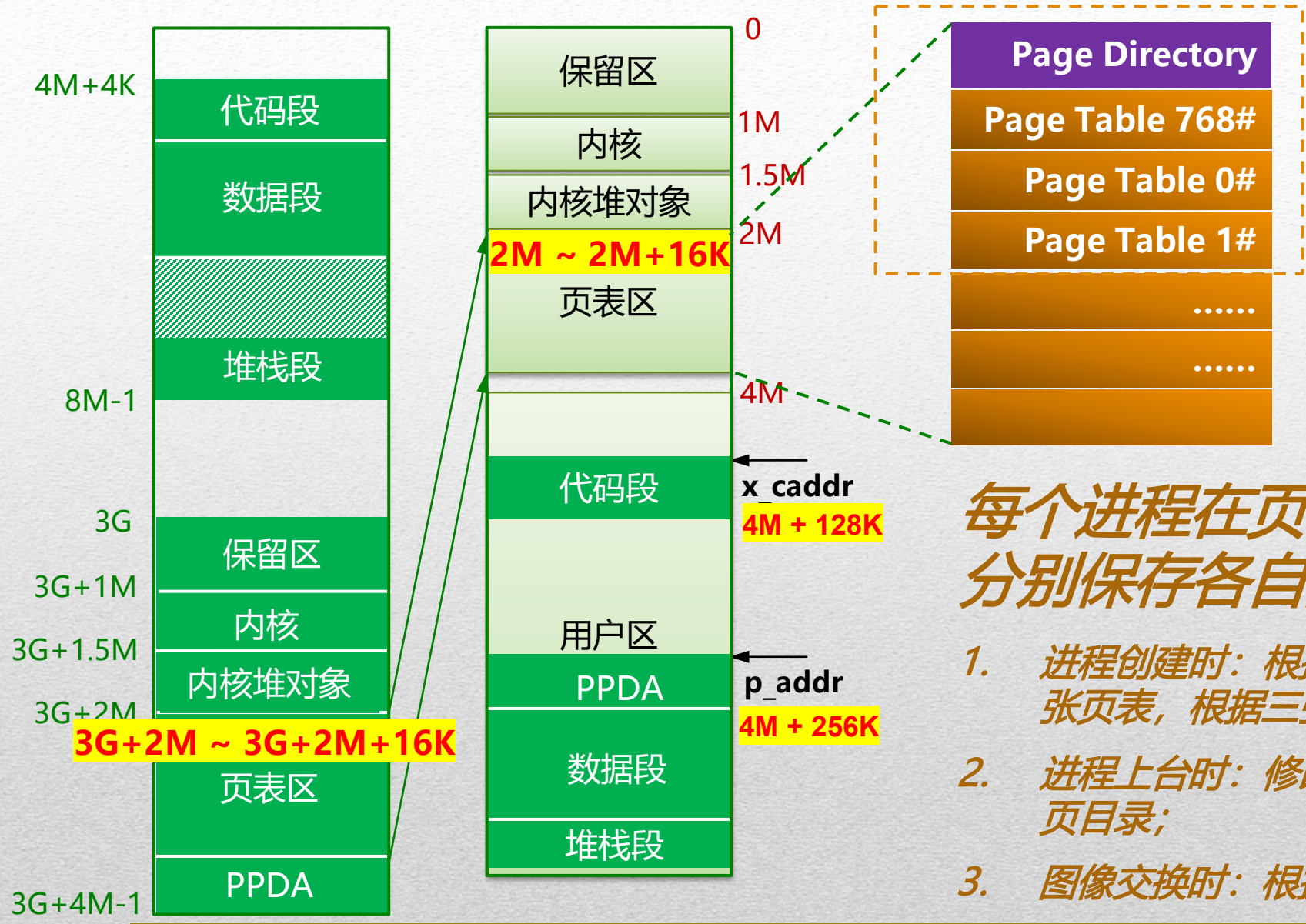
	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
	p_addr>>12+1		1	1	1
	.....	...	...	...	...
	p_addr>>12+1+m				
	.....	...	...	...	...
1023#	p_addr>>12+1+m+1		1	1	1

Page Table 768#

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....	...	...	...	...
1023#	p_addr>>12		0	1	1

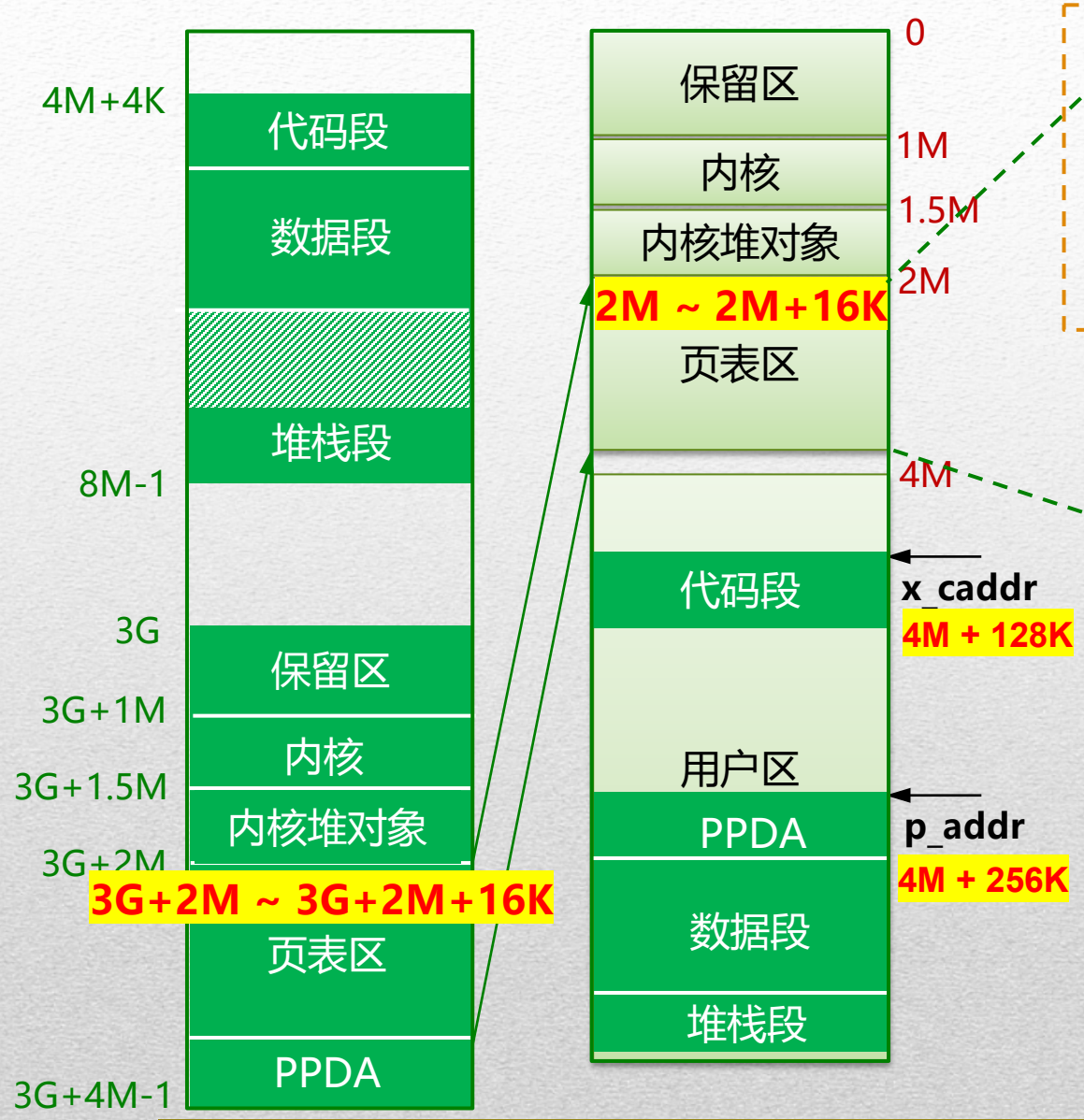
三张页表保存在哪里?

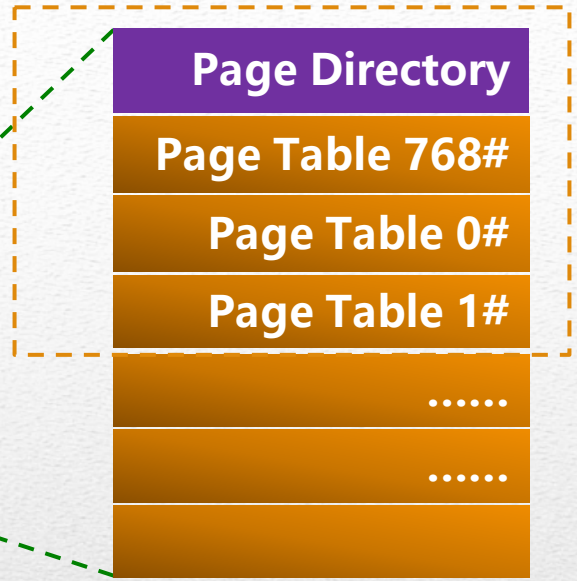
页目录保存在哪里?



每个进程在页表区占用4个页（框），分别保存各自的页表和页目录：

1. *进程创建时：根据p_addr和x_caddr的值写好三张页表，根据三张页表的位置设置好页目录；*
2. *进程上台时：修改CR3寄存器的值，指向该进程的页目录；*
3. *图像交换时：根据p_addr和x_caddr的值刷页表*



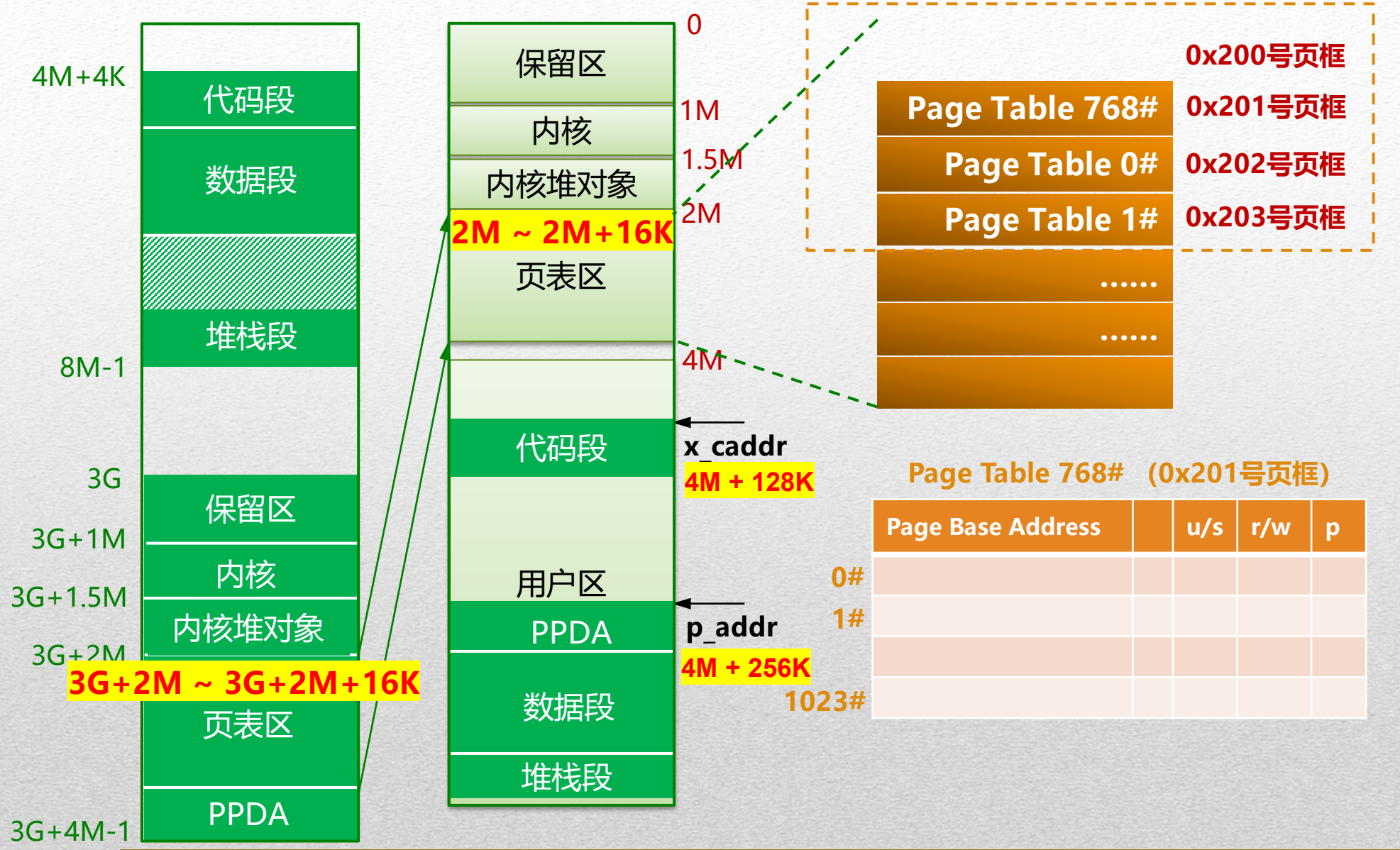


浪费空间

浪费时间

每个进程在页表区占用4个页（框），分别保存各自的页表和页目录：

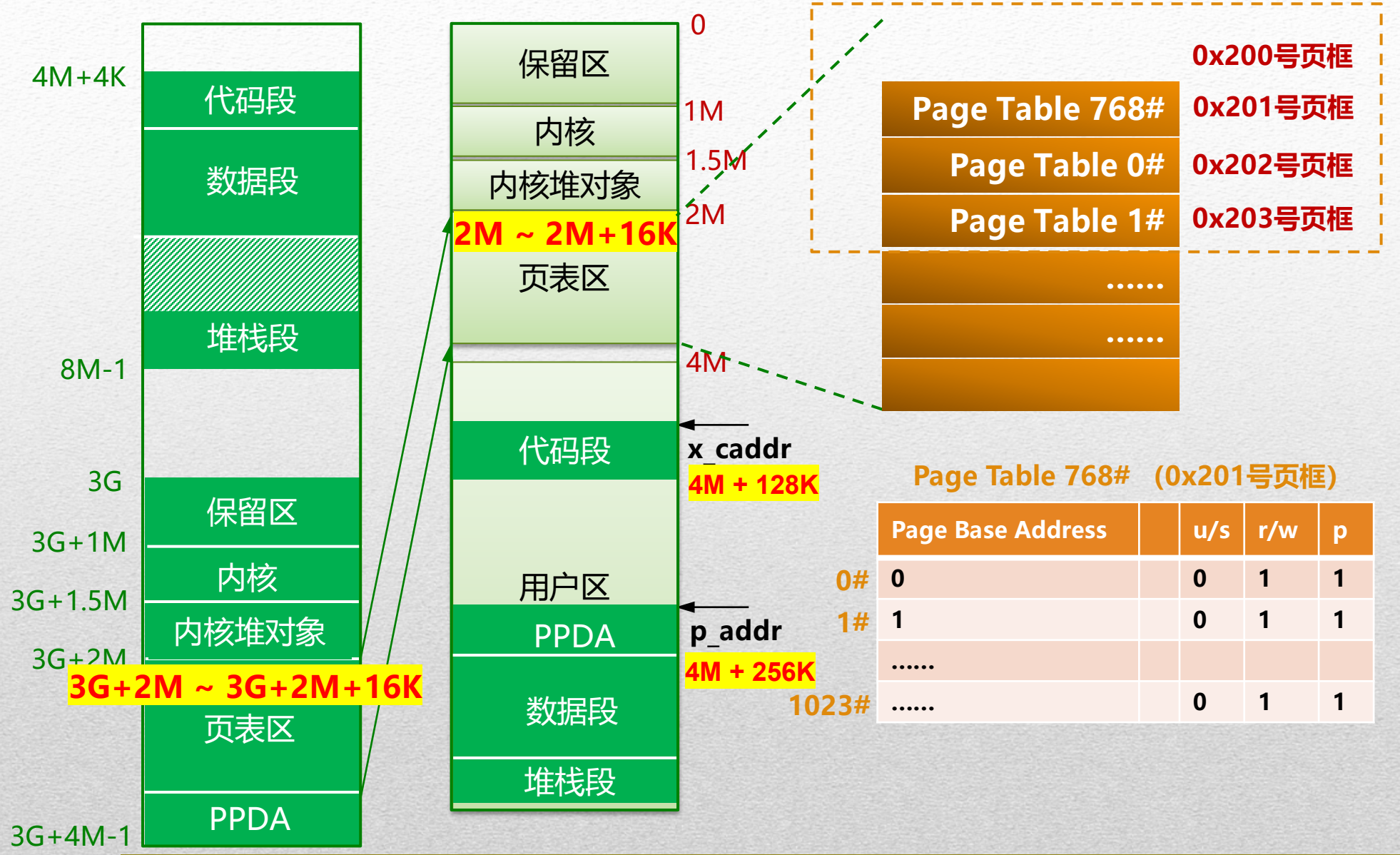
1. *进程创建时：根据p_addr和x_caddr的值写好三张页表，根据三张页表的位置设置好页目录；*
2. *进程上台时：修改CR3寄存器的值，指向该进程的页目录；*
3. *图像交换时：根据p_addr和x_caddr的值刷页表*



现运行进程的内核页表与用户页表

Page Table 768# (0x201号页框)

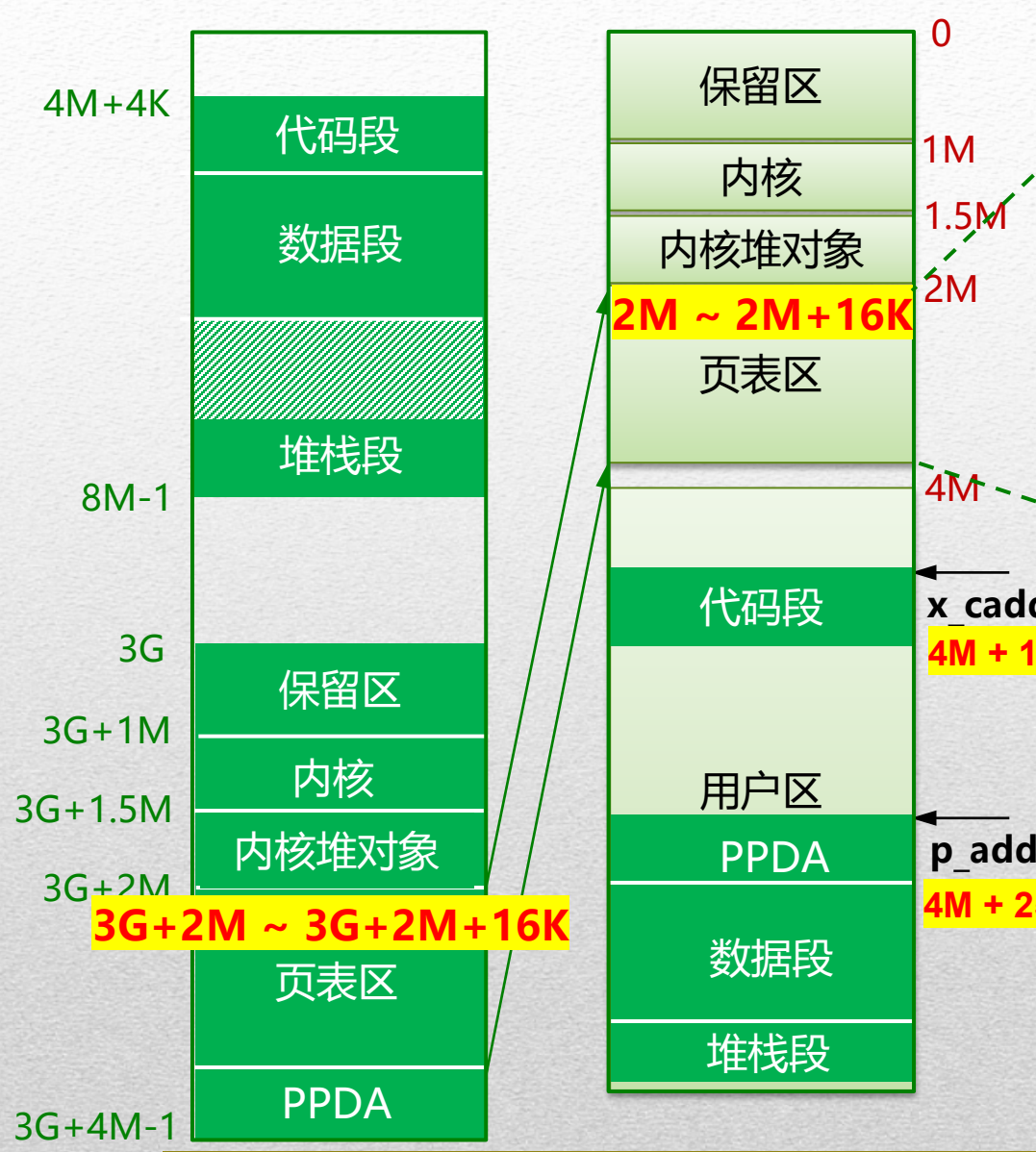
	Page Base Address		u/s	r/w	p
0#					
1#					
1023#					

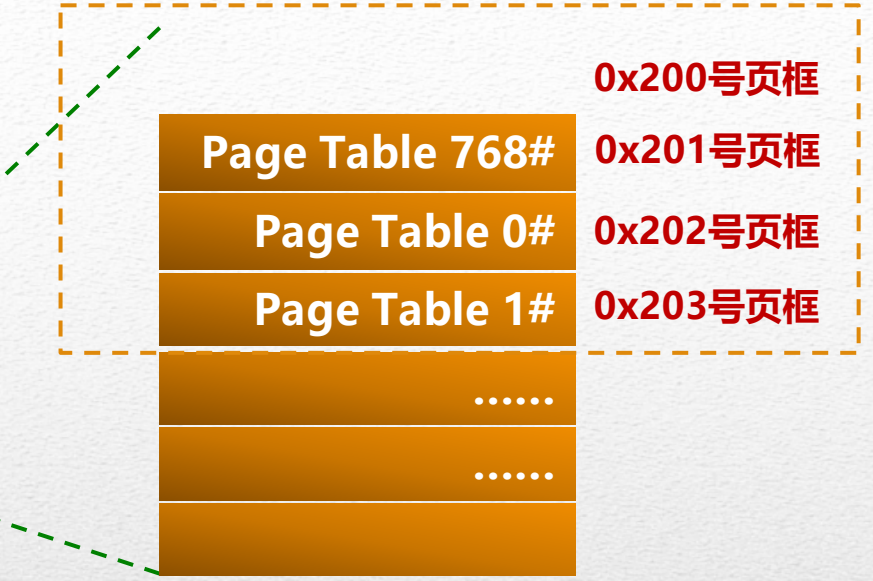


现运行进程的内核页表与用户页表

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	.....		0	1	1





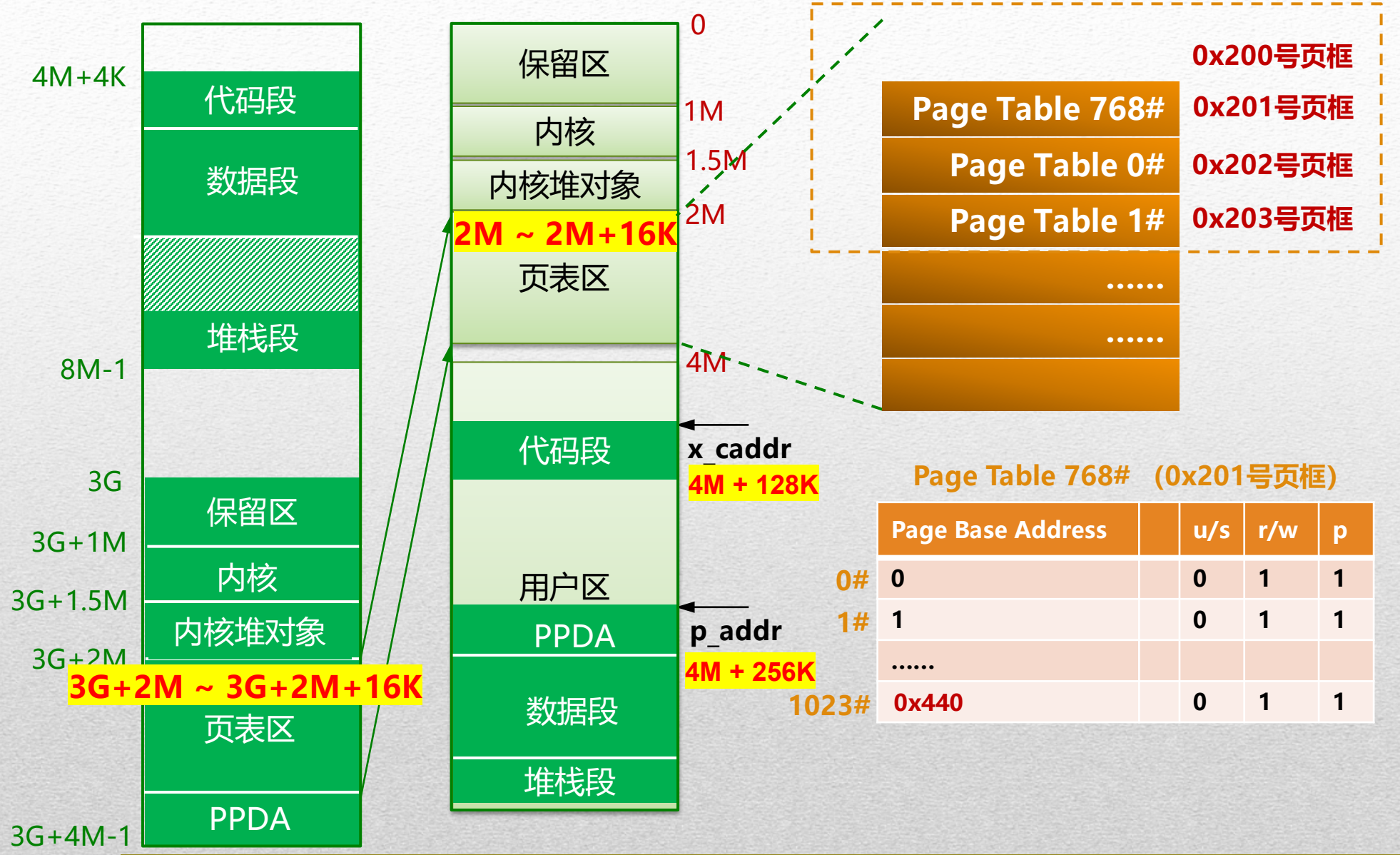
现运行进程的内核页表与用户页表

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	.....		0	1	1

s，内核

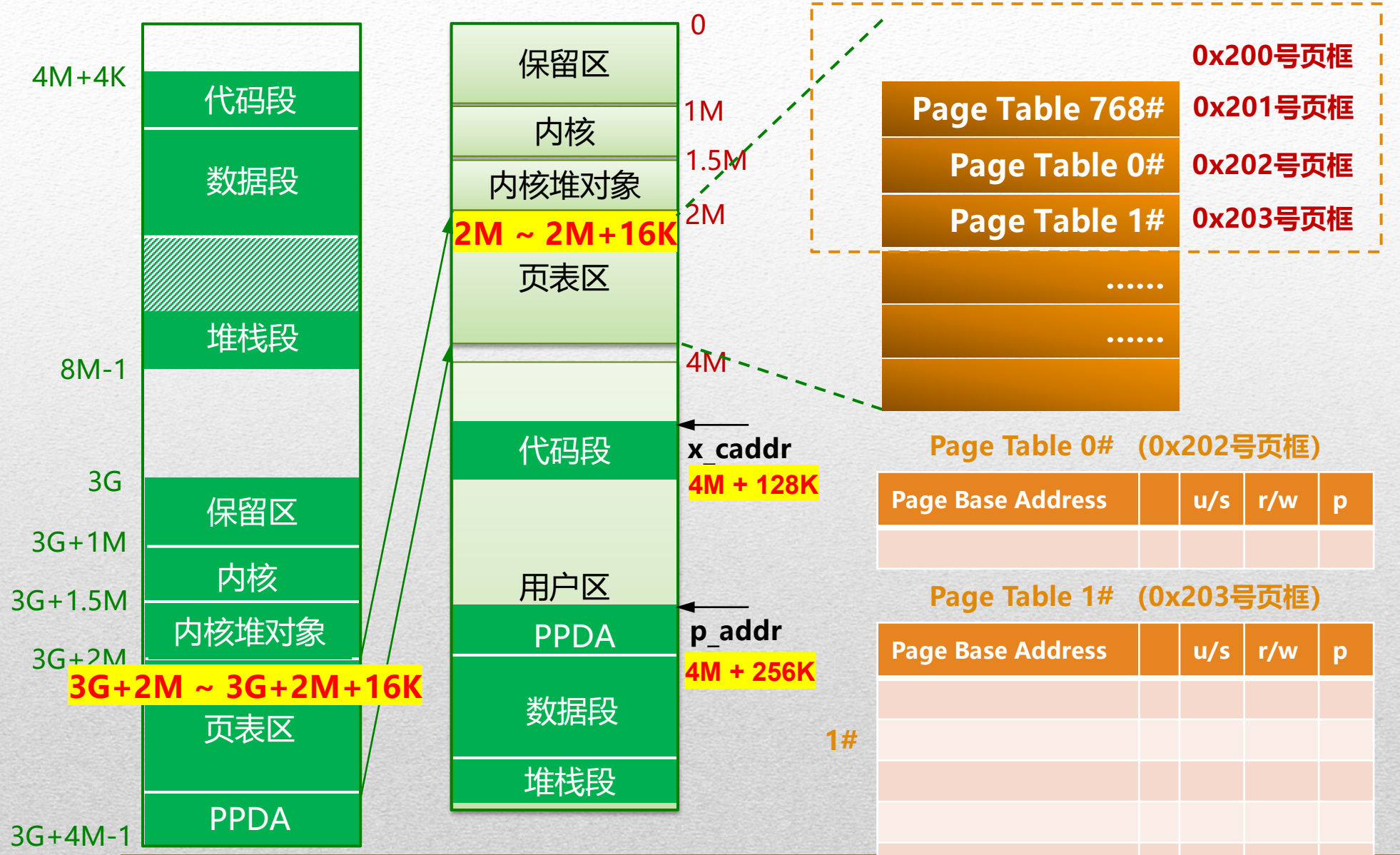
rw，读写



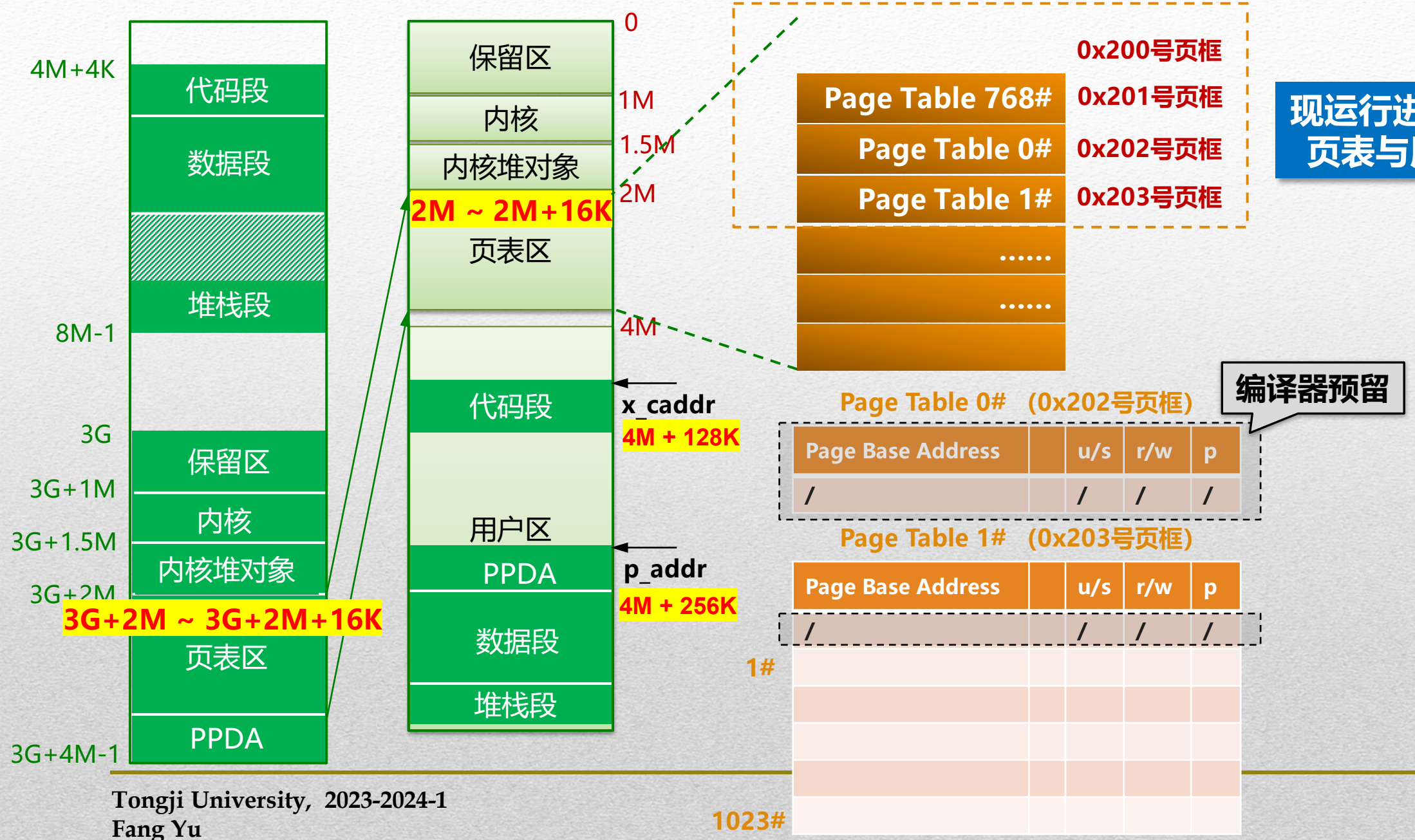
现运行进程的内核页表与用户页表

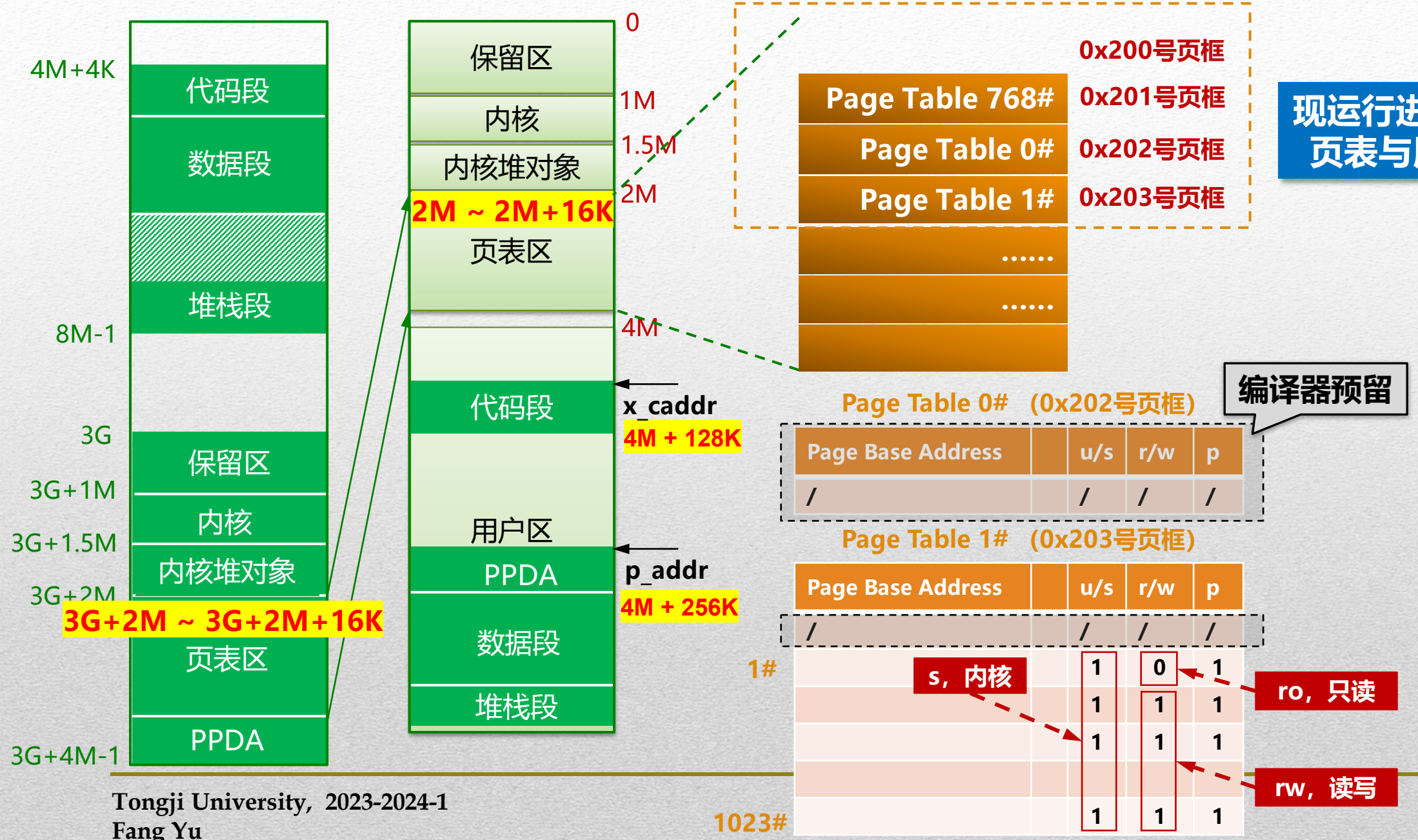
Page Table 768# (0x201号页框)

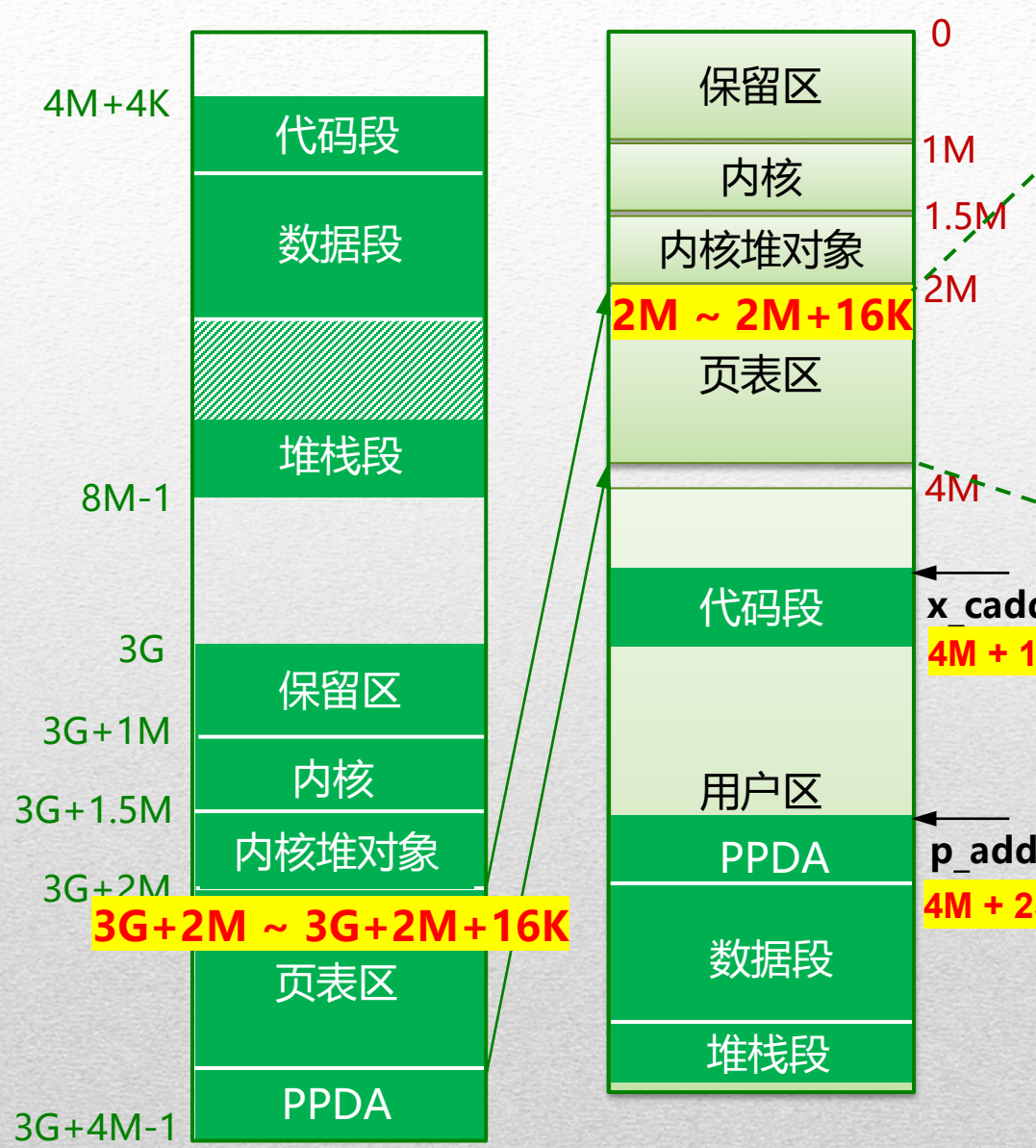
	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x440		0	1	1



现运行进程的内核页表与用户页表


4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
3G+2M ~ 3G+2M+16K
页表区
PPDA
3G+4M-1
0
保留区
1M
内核
1.5M
内核堆对象
2M
2M ~ 2M+16K
页表区
4M
代码段
x_caddr
4M + 128K
用户区
PPDA
p_addr
4M + 256K
数据段
堆栈段
Page Table 768#
Page Table 0#
Page Table 1#
.....
.....
0x200号页框
0x201号页框
0x202号页框
0x203号页框
现运行进程的内核页表与用户页表
编译器预留
Page Table 0# (0x202号页框)
Page Base Address
u/s
r/w
p
/
Page Table 1# (0x203号页框)
1#
1023#


4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
3G+2M ~ 3G+2M+16K
页表区
PPDA
3G+4M-1
0
保留区
1M
内核
1.5M
内核堆对象
2M
2M ~ 2M+16K
页表区
4M
代码段
x_caddr
4M + 128K
用户区
PPDA
p_addr
4M + 256K
数据段
堆栈段
0x200号页框
Page Table 768#
0x201号页框
Page Table 0#
0x202号页框
Page Table 1#
0x203号页框
.....
.....
现运行进程的内核页表与用户页表
编译器预留
Page Table 0# (0x202号页框)
Page Base Address
u/s
r/w
p
/
/
/
/
Page Table 1# (0x203号页框)
Page Base Address
u/s
r/w
p
/
/
/
/
1#
s，内核
1
0
1
1
1
1
1
1
1
1
1
1
ro，只读
rw，读写
1023#



现运行进程的内核页表与用户页表

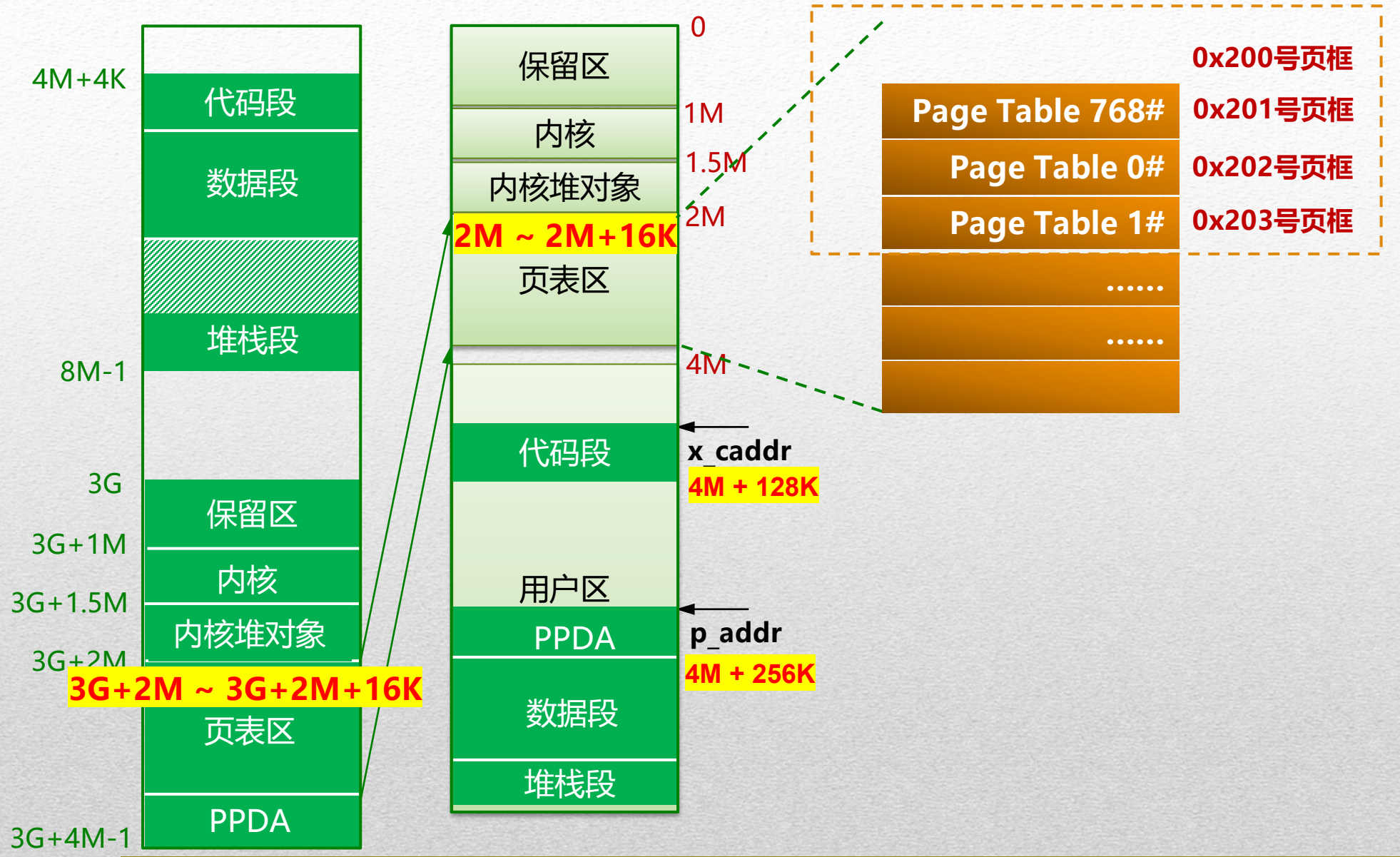
编译器预留

Page Table 0# (0x202号页框)

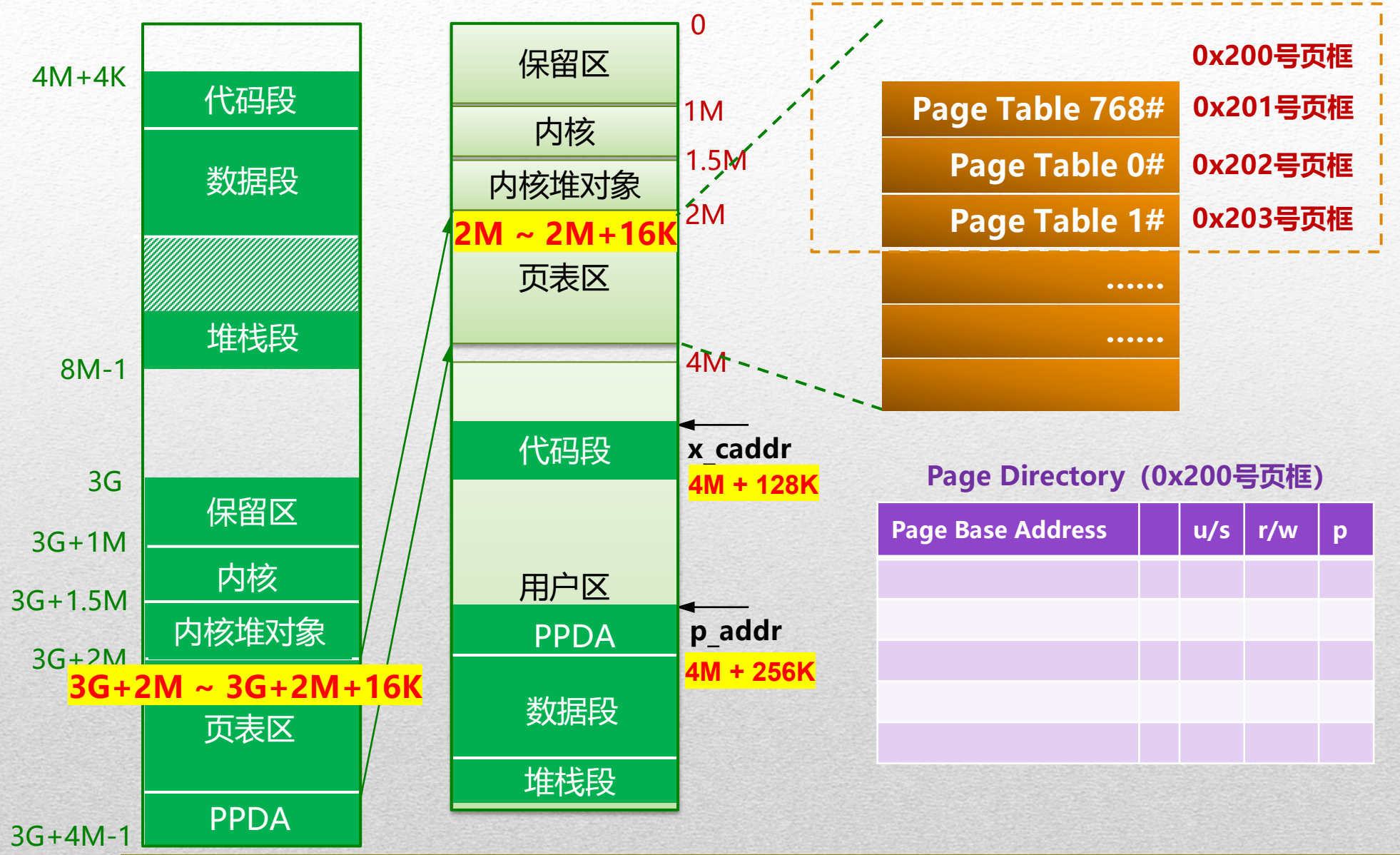
Page Base Address		u/s	r/w	p
/		/	/	/

Page Table 1# (0x203号页框)

	Page Base Address		u/s	r/w	p
	/		/	/	/
1#	0x420		1	0	1
	0x441		1	1	1
	0x442		1	1	1
1023#	0x443		1	1	1



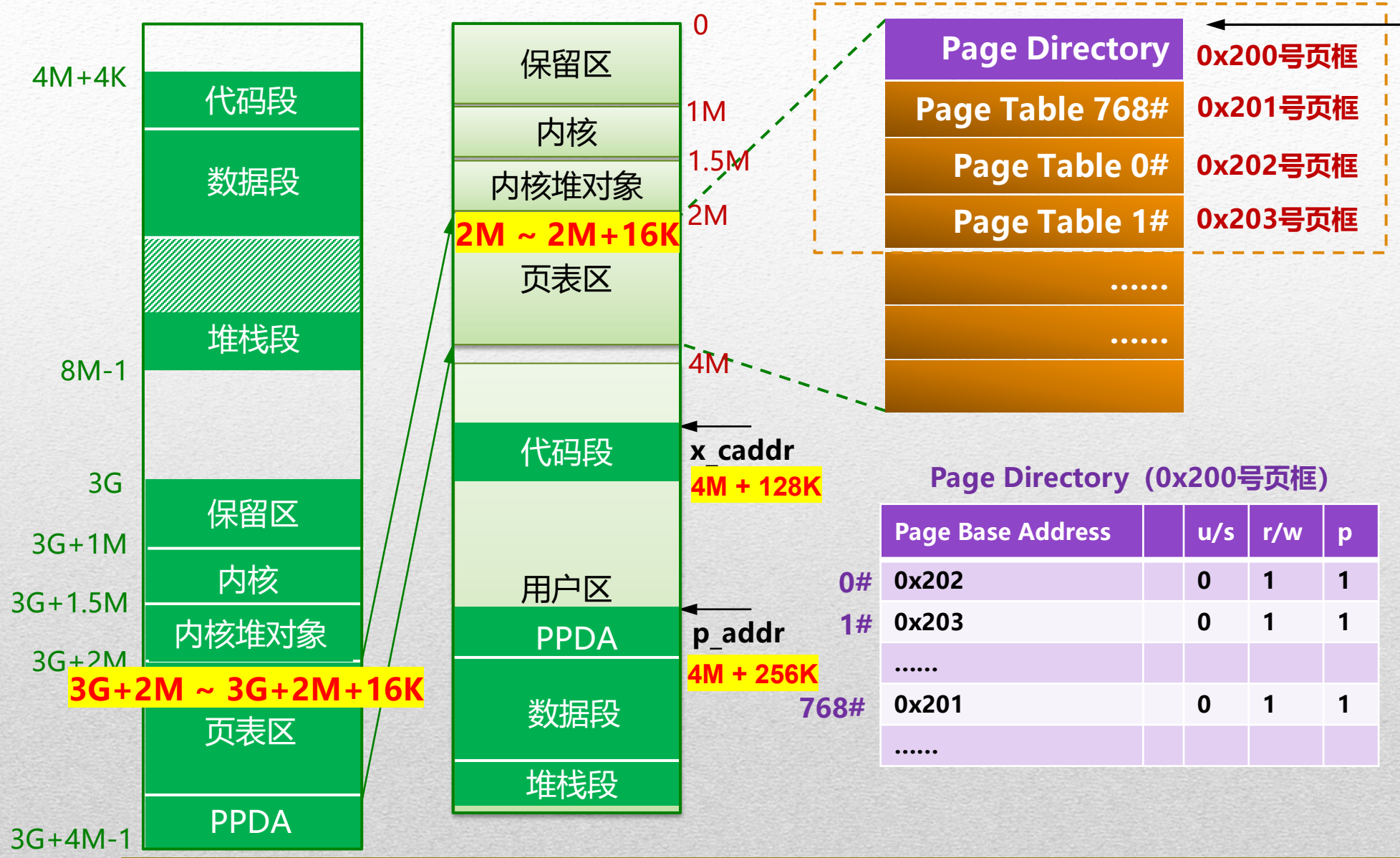
现运行进程的内核页表与用户页表



现运行进程的内核页表与用户页表

Page Directory（0x200号页框）

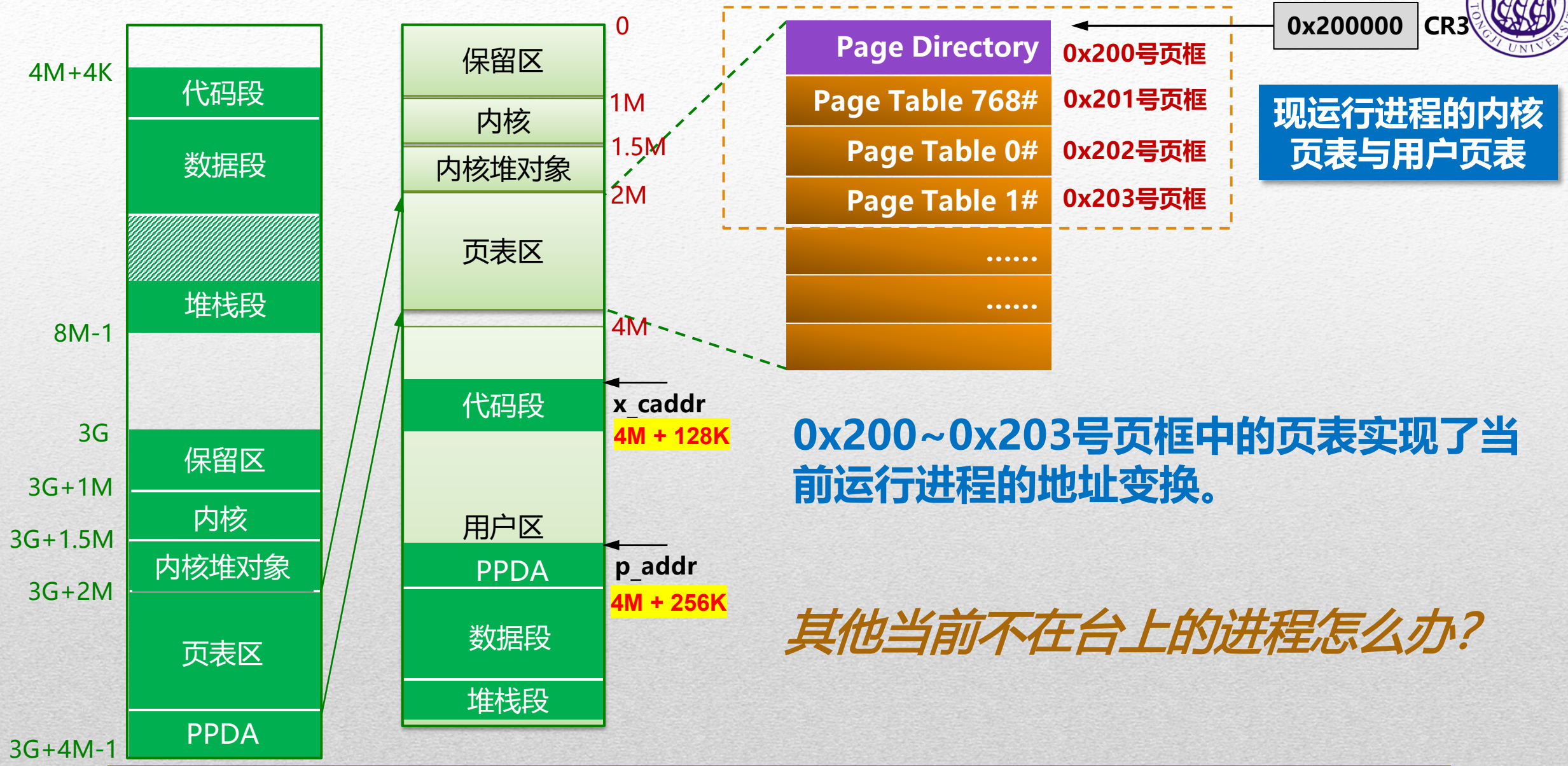
Page Base Address		u/s	r/w	p



现运行进程的内核页表与用户页表

Page Directory（0x200号页框）

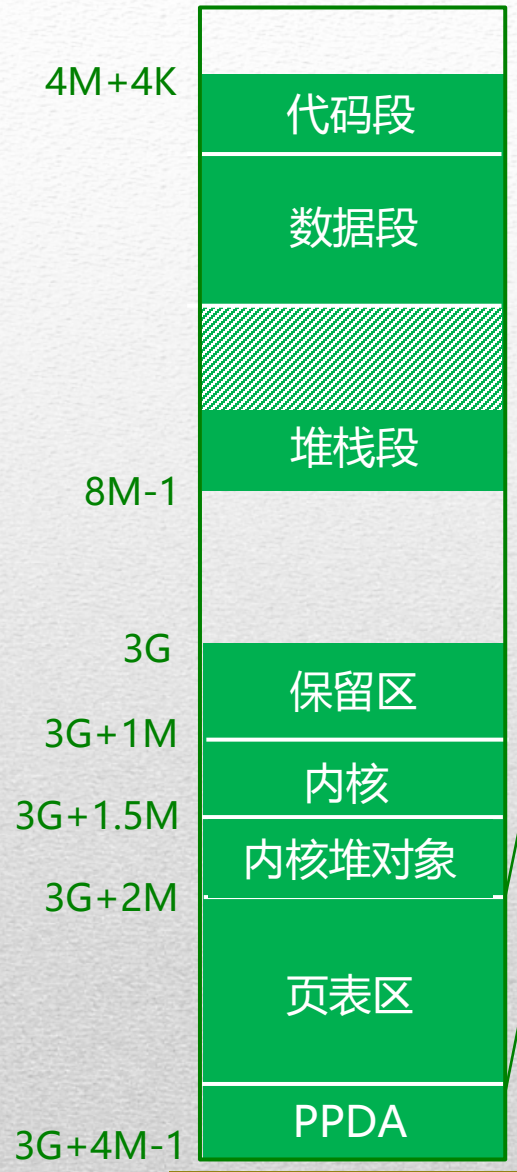
	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	.....				
768#	0x201		0	1	1
	.....				



现运行进程的内核页表与用户页表

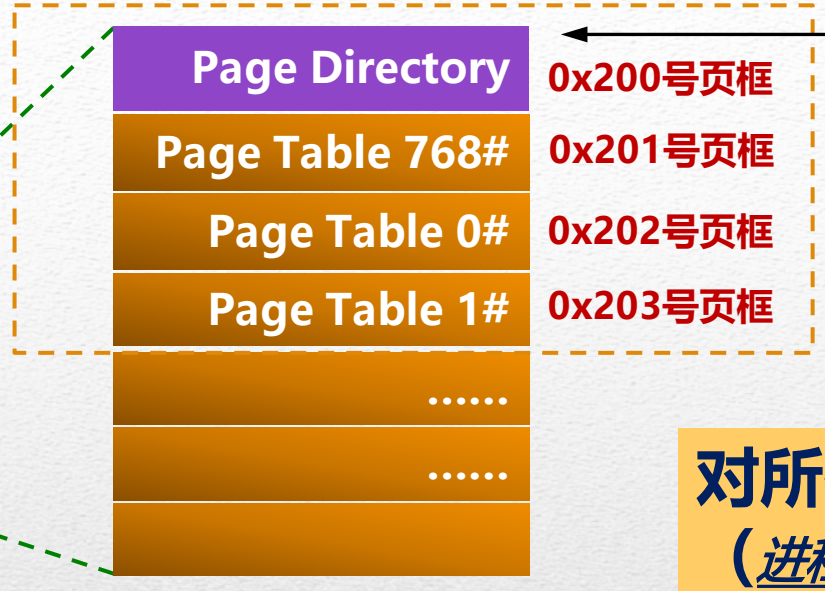
0x200~0x203号页框中的页表实现了当前运行进程的地址变换。

其他当前不在台上的进程怎么办?



x_caddr
4M + 128K

p_addr
4M + 256K



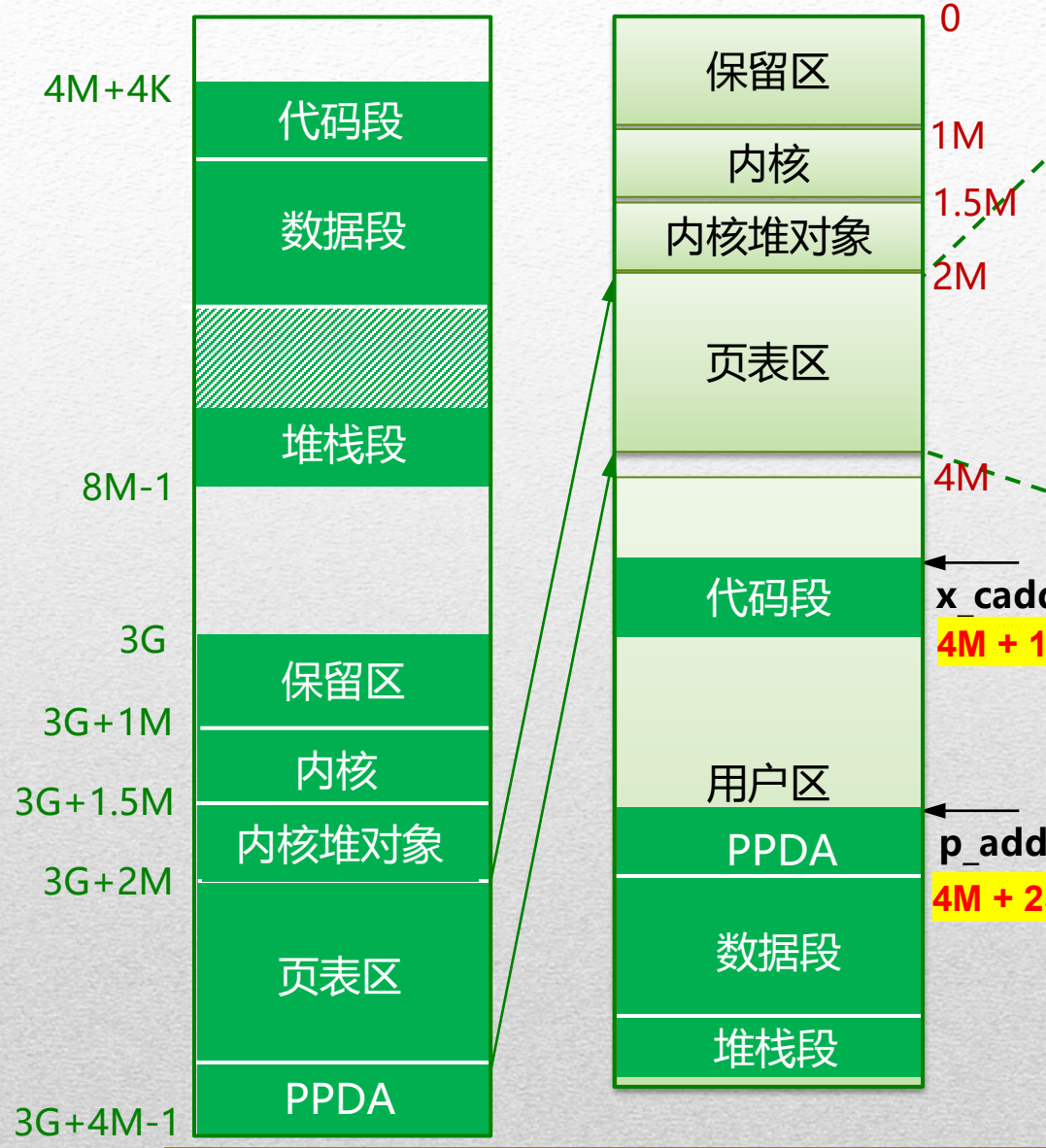
0x200000 CR3

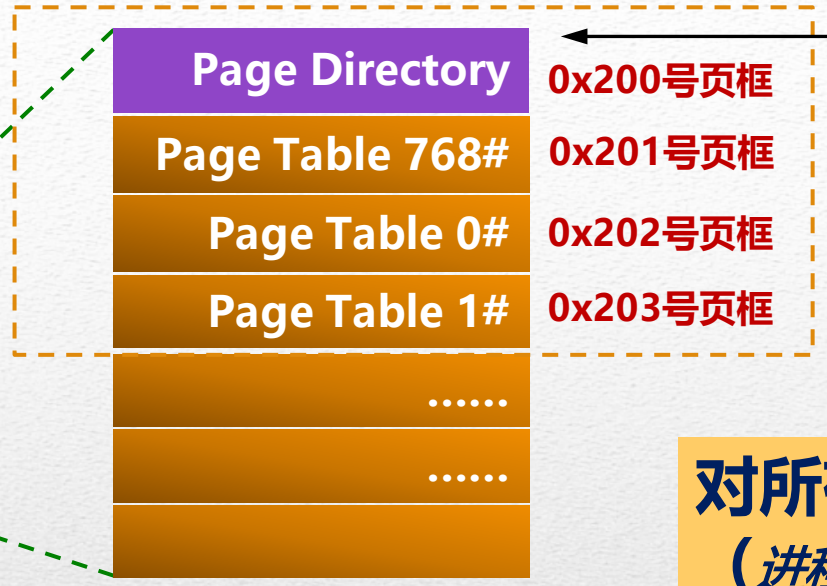
现运行进程的内核页表与用户页表

对所有进程都是一样的
(*进程切换调度时不需修改*)

Page Directory（0x200号页框）

	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	.....				
768#	0x201		0	1	1
	.....				





0x200000 CR3

现运行进程的内核页表与用户页表

对所有进程都是一样的

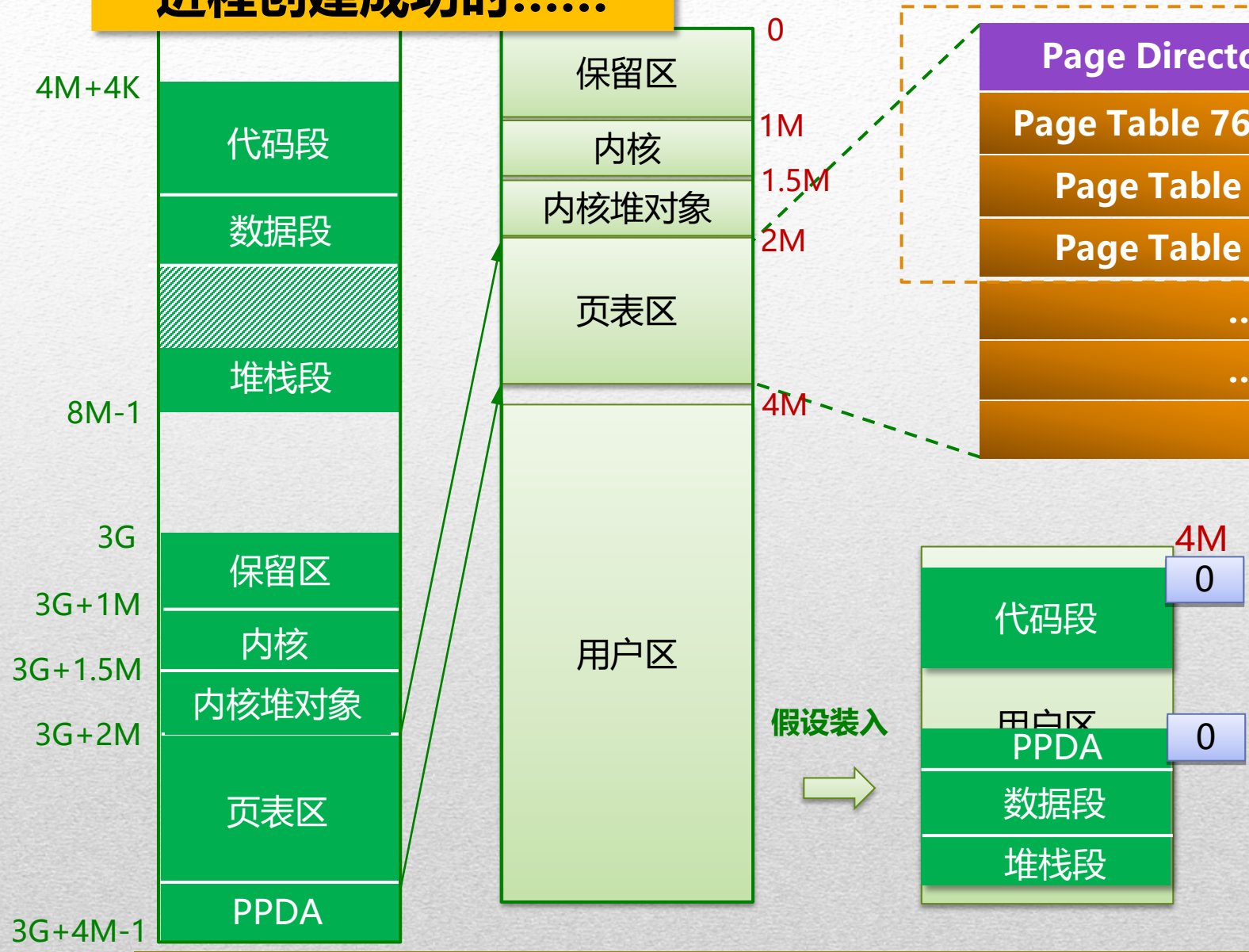
(*进程切换调度时不需修改*)

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	.....		0	1	1

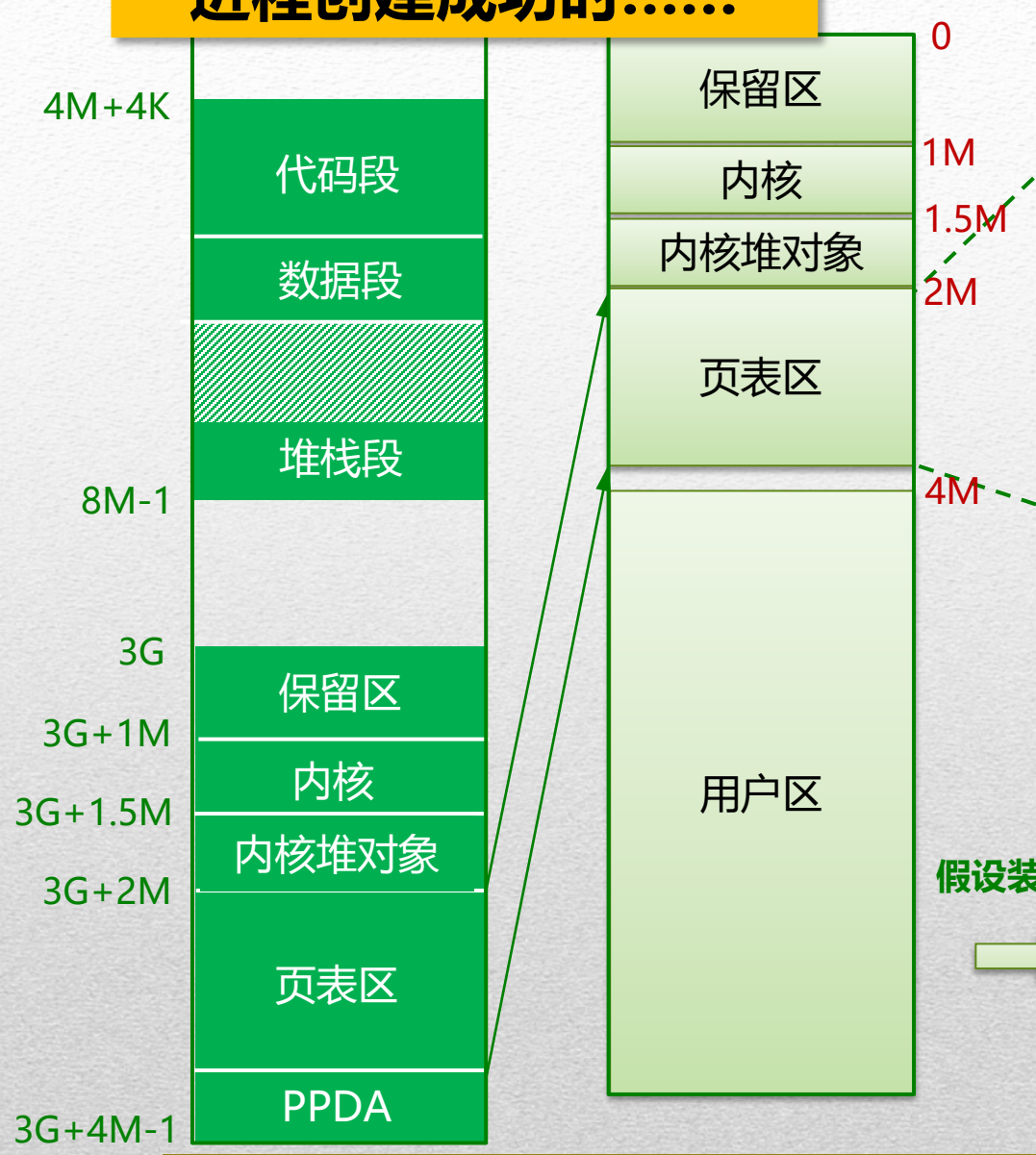
每个页目录和核心页表进程共用，节省空间

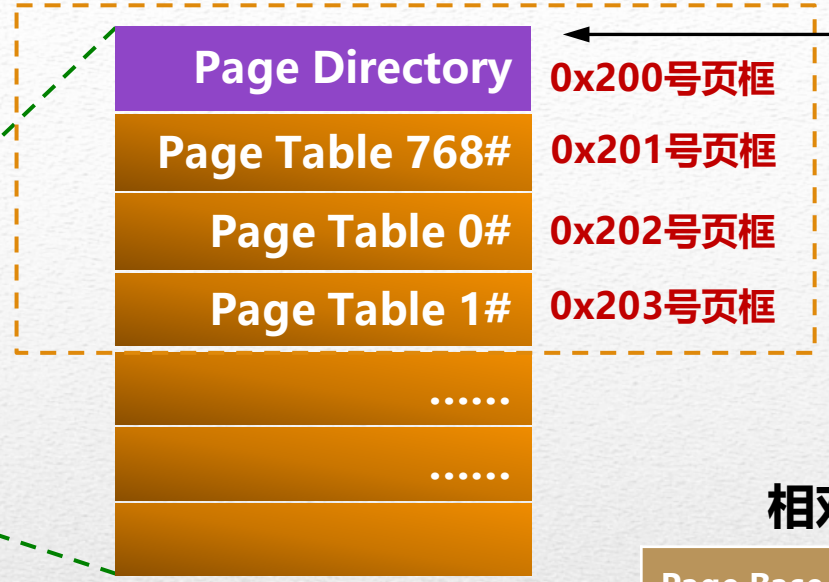
进程创建成功时……



现运行进程的内核页表与用户页表

进程创建成功时……





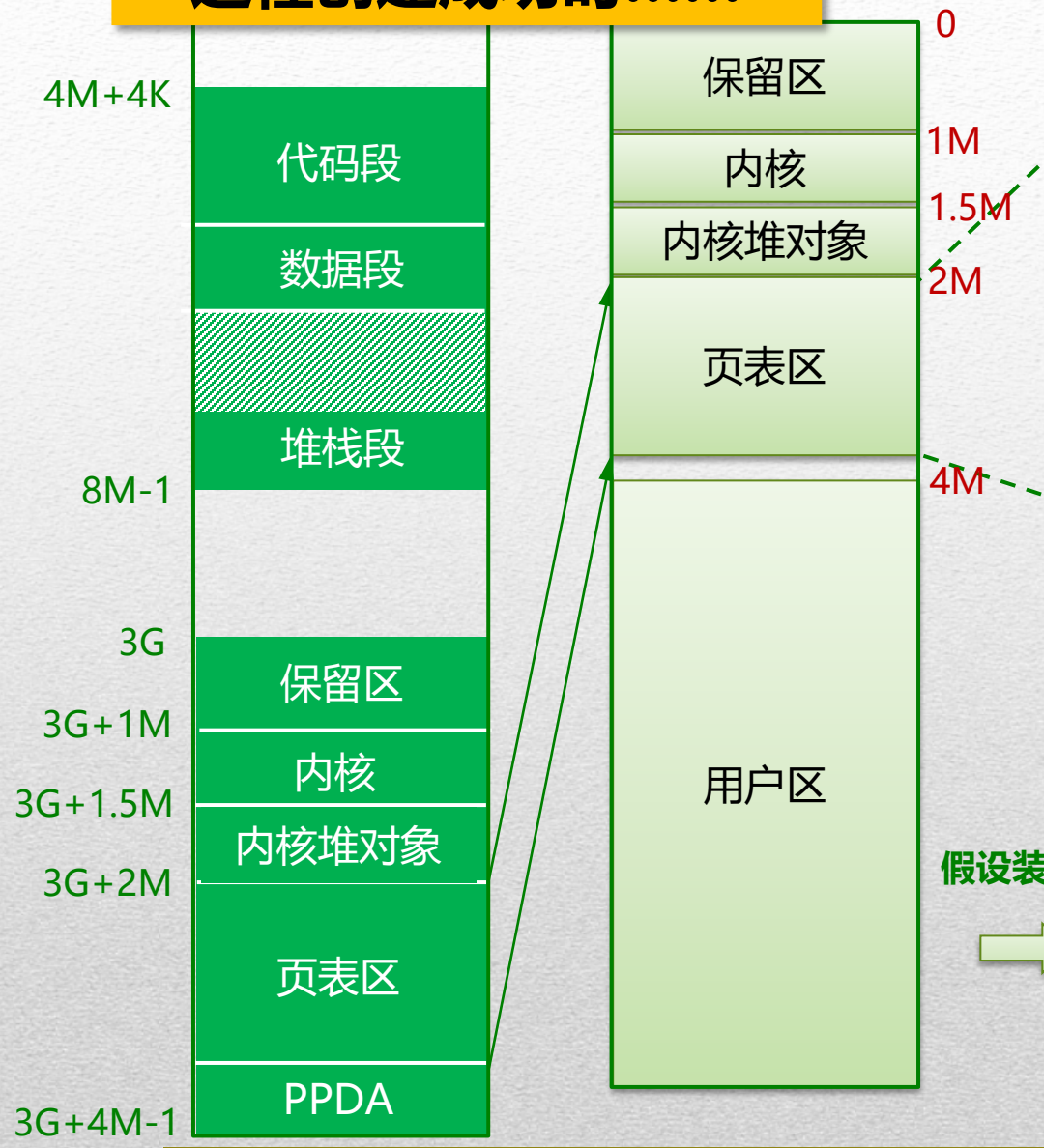
现运行进程的内核页表与用户页表

假设装入

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#	1		1	1	1
	全0				
2047#	2		1	1	1

进程创建成功时……



现运行进程的内核页表与用户页表

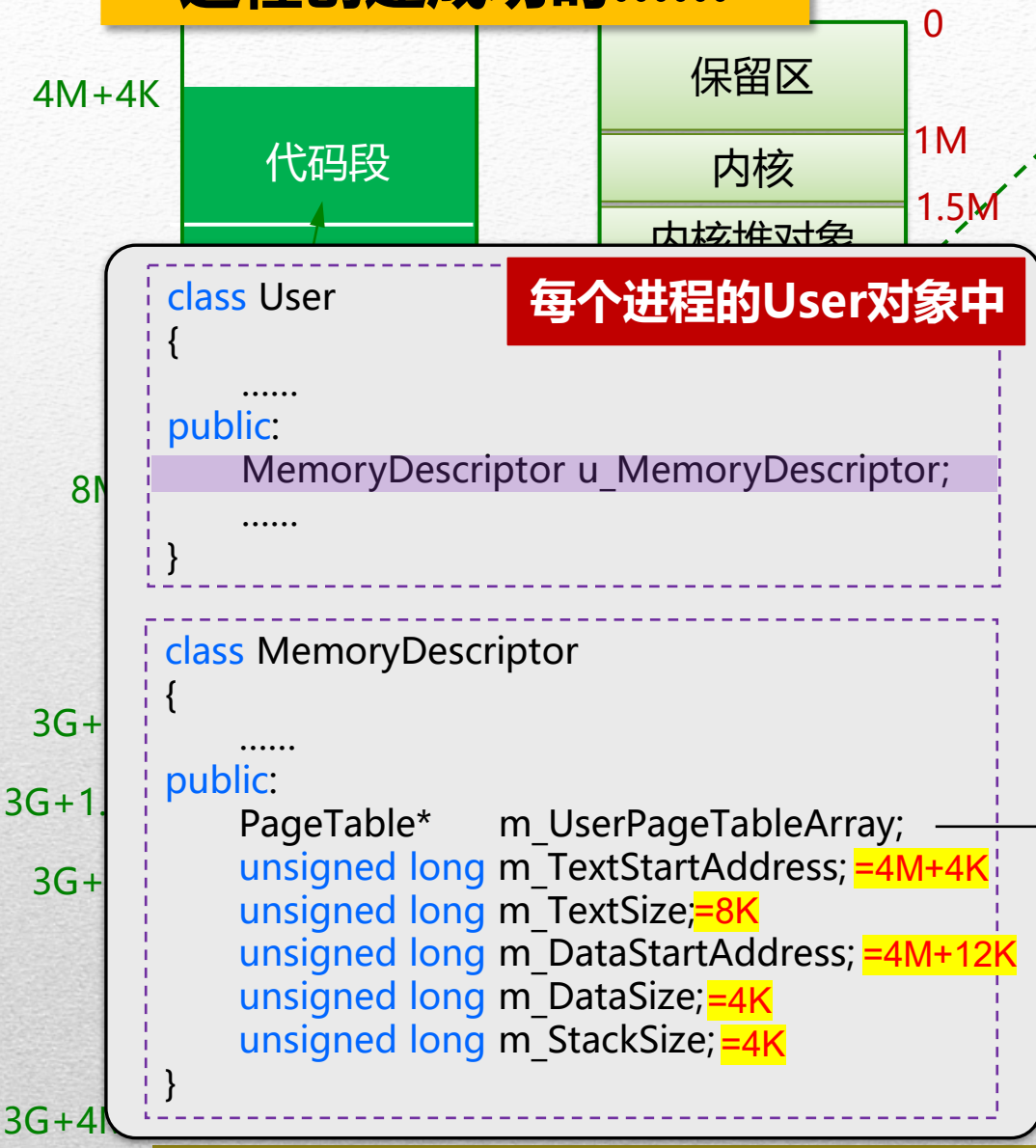
占用两个连续页框

假设装入

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#	1		1	1	1
	全0				
2047#	2		1	1	1

进程创建成功时……



每个进程的User对象中

```
class User
{
        ……
public:
        MemoryDescriptor u_MemoryDescriptor;
        ……
}
```

```
class MemoryDescriptor
{
        ……
public:
        PageTable*      m_UserPageTableArray;
        unsigned long m_TextStartAddress; =4M+4K
        unsigned long m_TextSize;=8K
        unsigned long m_DataStartAddress; =4M+12K
        unsigned long m_DataSize;=4K
        unsigned long m_StackSize; =4K
}
```

现运行进程的内核页表与用户页表

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#	1		1	1	1
	全0				
2047#	2		1	1	1

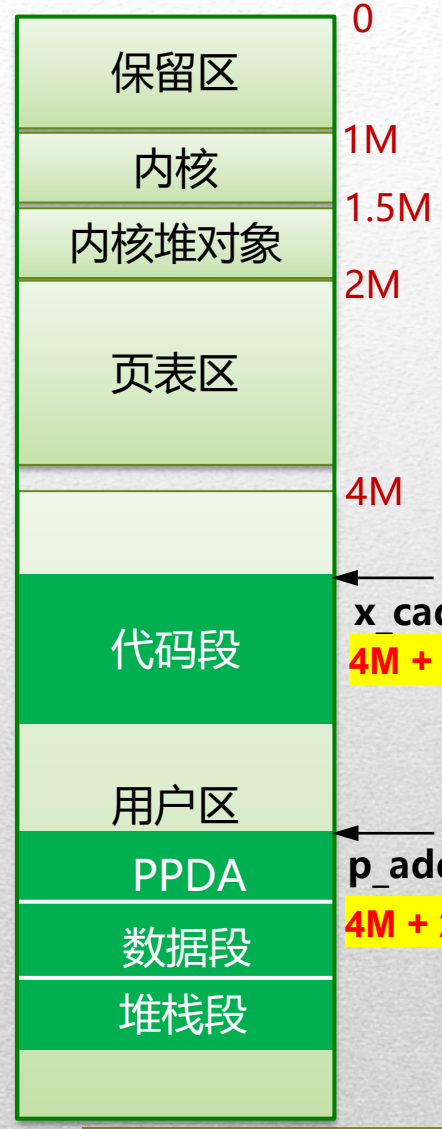
进程图像交换时……

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#	1		1	1	1
	全0				
2047#	2		1	1	1

每次换进/换出，只需修改p_addr和x_caddr，无需刷新页表，节省时间

进程被调度上台时……



Page Directory（0x200号页框）

	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	……				
768#	0x201		0	1	1
	……				

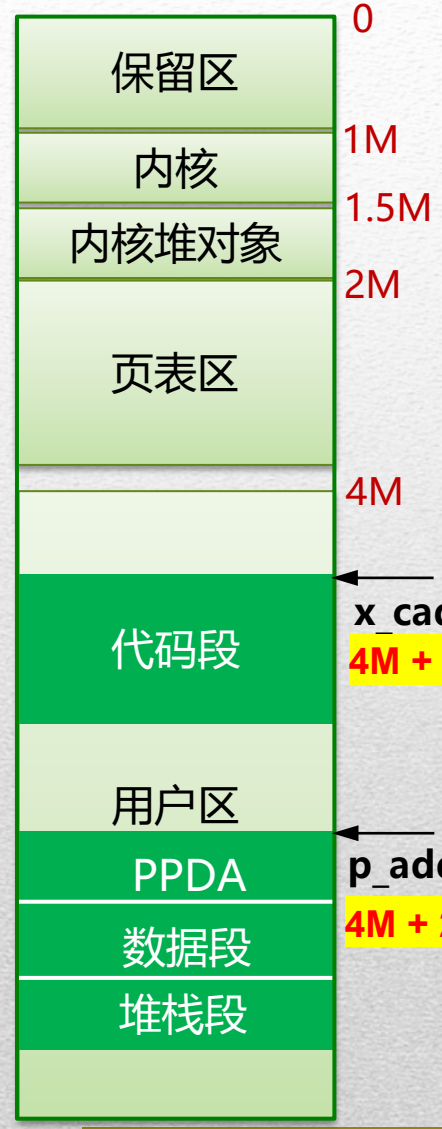
Page Table 768#（0x201号页框）

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1023#	……		0	1	1

对所有进程都是一样的

（*进程切换调度时不需修改*）

进程被调度上台时……



Page Directory（0x200号页框）

	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	……				
768#	0x201		0	1	1
	……				

①. p_addr>>12，写入

Page Table 768#（0x201号页框）

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1023#	0x440		0	1	1

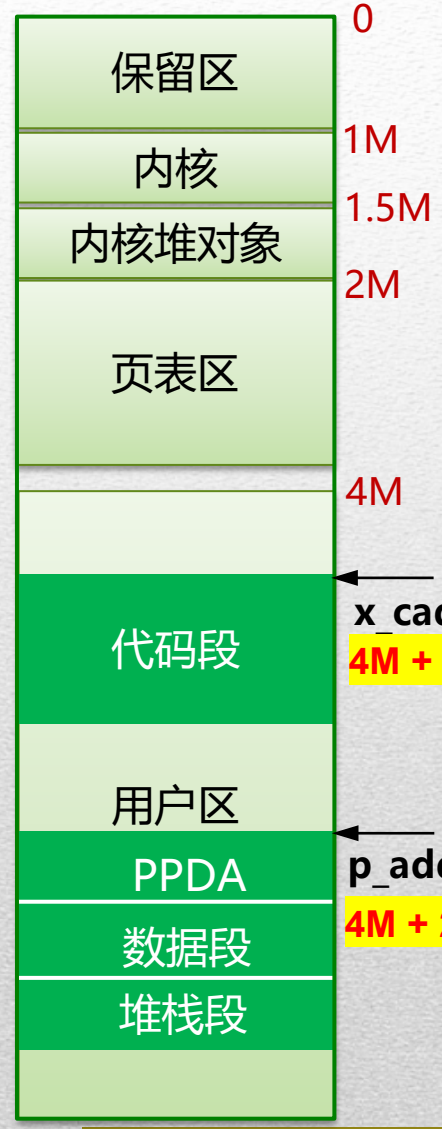
Page Table 0#（0x202号页框）

Page Base Address		u/s	r/w	p

Page Table 1#（0x203号页框）

	Page Base Address		u/s	r/w	p
1#					
1023#					

进程被调度上台时……



x_caddr
4M + 128K

p_addr
4M + 256K

Page Directory（0x200号页框）

	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	……				
768#	0x201		0	1	1
	……				

Page Table 768# （0x201号页框）

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1023#	0x440		0	1	1

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#					
	全0				
2047#	2		1	1	1

②. if r/w==0, x_caddr>>12，加上PBA

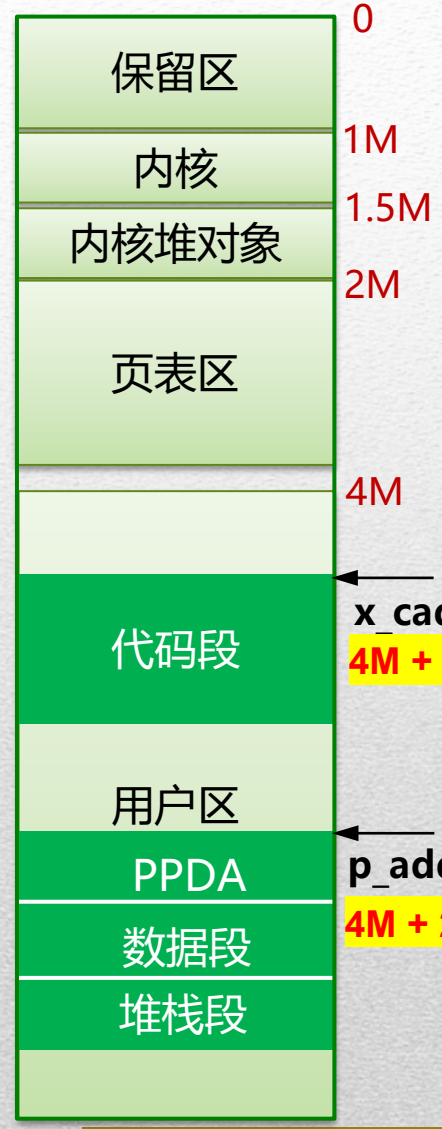
Page Table 0# （0x202号页框）

Page Base Address		u/s	r/w	p

Page Table 1# （0x203号页框）

	Page Base Address		u/s	r/w	p
1#					
	0x420		1	0	1
	0x421		1	0	1
1023#					

进程被调度上台时……



x_caddr
4M + 128K

p_addr
4M + 256K

Page Directory（0x200号页框）

	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	……				
768#	0x201		0	1	1
	……				

Page Table 768#（0x201号页框）

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1023#	0x440		0	1	1

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#	1		1	1	1
	全0				
2047#	2		1	1	1

Page Table 0#（0x202号页框）

Page Base Address		u/s	r/w	p

Page Table 1#（0x203号页框）

	Page Base Address		u/s	r/w	p
1#					
	0x420		1	0	1
	0x421		1	0	1
	0x441		1	1	1
1023#	0x442		1	1	1

③. if r/w==1, p_addr>>12，加上PBA

如果进程执行 程序地址[4M+7k] 处一条指令

0x200000 CR3

4M+7K

G	M	K
00	00 0000 01	00 0000 0001	11 00 0000 0000

Page Directory（0x200号页框）

Page Base Address		u/s	r/w	p
0x202		0	1	1
0x203		0	1	1
.....				
0x201		0	1	1
.....				

Page Table 1# （0x203号页框）

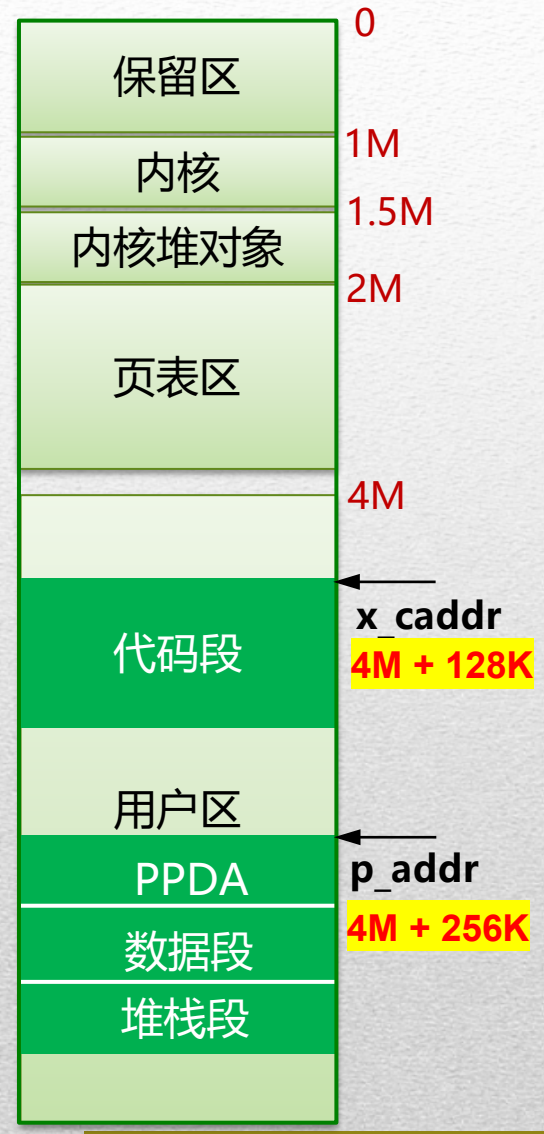
Page Base Address		u/s	r/w	p
0x420		1	0	1
0x421		1	0	1
0x441		1	1	1
0x442		1	1	1

4M+131K

00 00 0000 01 00 0010 0000	11 00 0000 0000

另一种计算方法： 4M+4K ⟶ 4M+128K 4M+7K ⟶ 4M+131K

如果该指令为 inc [4M+14k]



0x200000 CR3

4M+14K

G	M	K
00 00 0000 01	00 0000 0011	10 00 0000 0000

Page Directory（0x200号页框）

Page Base Address		u/s	r/w	p
0x202		0	1	1
0x203		0	1	1
.....				
0x201		0	1	1
.....				

Page Table 1# （0x203号页框）

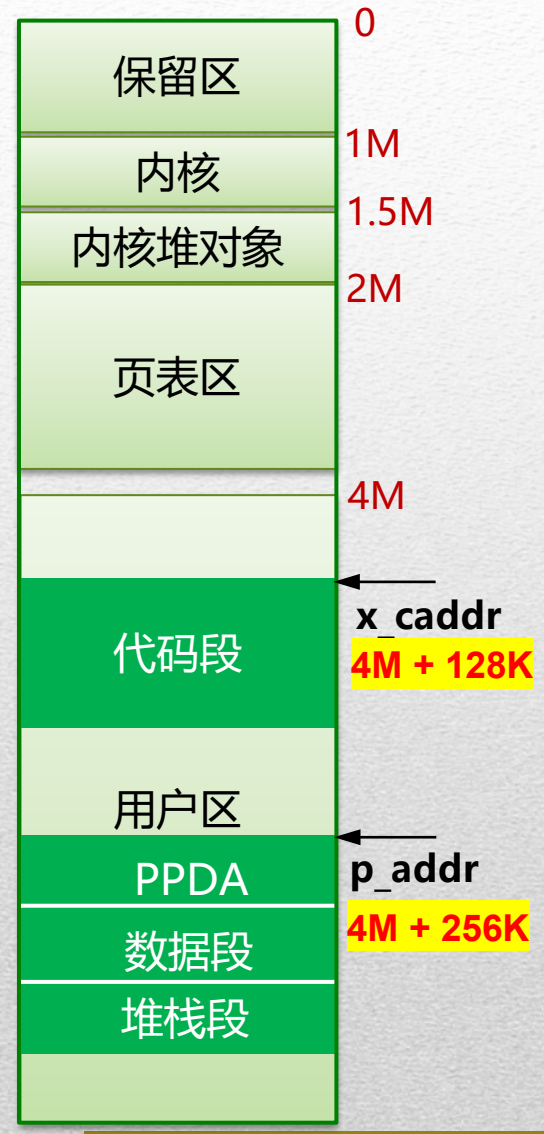
Page Base Address		u/s	r/w	p
0x420		1	0	1
0x421		1	0	1
0x441		1	1	1
0x442		1	1	1

4M+262K

00 00 0000 01 00 0100 0001	10 00 0000 0000

另一种计算方法：4M+12K → 4M+260K　4M+14K → 4M+262K

如果系统中还有另一个进程



有一样的相对虚实地址映射表：

相对虚实地址映射表

	Page Base Address		u/s	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	0	1
1027#	1		1	1	1
	全0				
2047#	2		1	1	1

说明两个进程有完全一样的程序地址

如果系统中还有另一个进程

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x440		0	1	1

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x410		0	1	1

Page Table 1# (0x203号页框)

	Page Base Address		u/s	r/w	p
1#					
	0x420		1	0	1
	0x421		1	0	1
	0x441		1	1	1
1023#	0x442		1	1	1

Page Table 1# (0x203号页框)

	Page Base Address		u/s	r/w	p
1#					
	0x420		1	0	1
	0x421		1	0	1
	0x411		1	1	1
1023#	0x412		1	1	1

说明两个进程共享代码段部分（代码中的逻辑地址相同）

4M+7K ⟶ 4M+131K：代码段部分逻辑地址经地址变换后指向相同的物理地址

如果系统中还有另一个进程

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x440		0	1	1

Page Table 768# (0x201号页框)

	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x410		0	1	1

4M + 64K

Page Table 1# (0x203号页框)

	Page Base Address		u/s	r/w	p
1#					
	0x420		1	0	1
	0x421		1	0	1
	0x441		1	1	1
1023#	0x442		1	1	1

Page Table 1# (0x203号页框)

	Page Base Address		u/s	r/w	p
1#					
	0x420		1	0	1
	0x421		1	0	1
	0x411		1	1	1
1023#	0x412		1	1	1

说明两个进程有各自的进程图像可交换部分

4M+14K → 4M+262K vs 4M+70K：数据段部分逻辑地址经地址变换后指向不同的物理地址

本节小结：

1 UNIX V6++中进程核心态与用户态下的逻辑地址空间

2 UNIX V6++中进程核心态与用户态下的物理地址空间

3 UNIX V6++中利用两级页表实现的地址变换过程

4 UNIX V6++如何获得现运行进程完整的图像