第三章

存储管理

方　钰

主 要 内 容

3.1 存储管理的主要任务

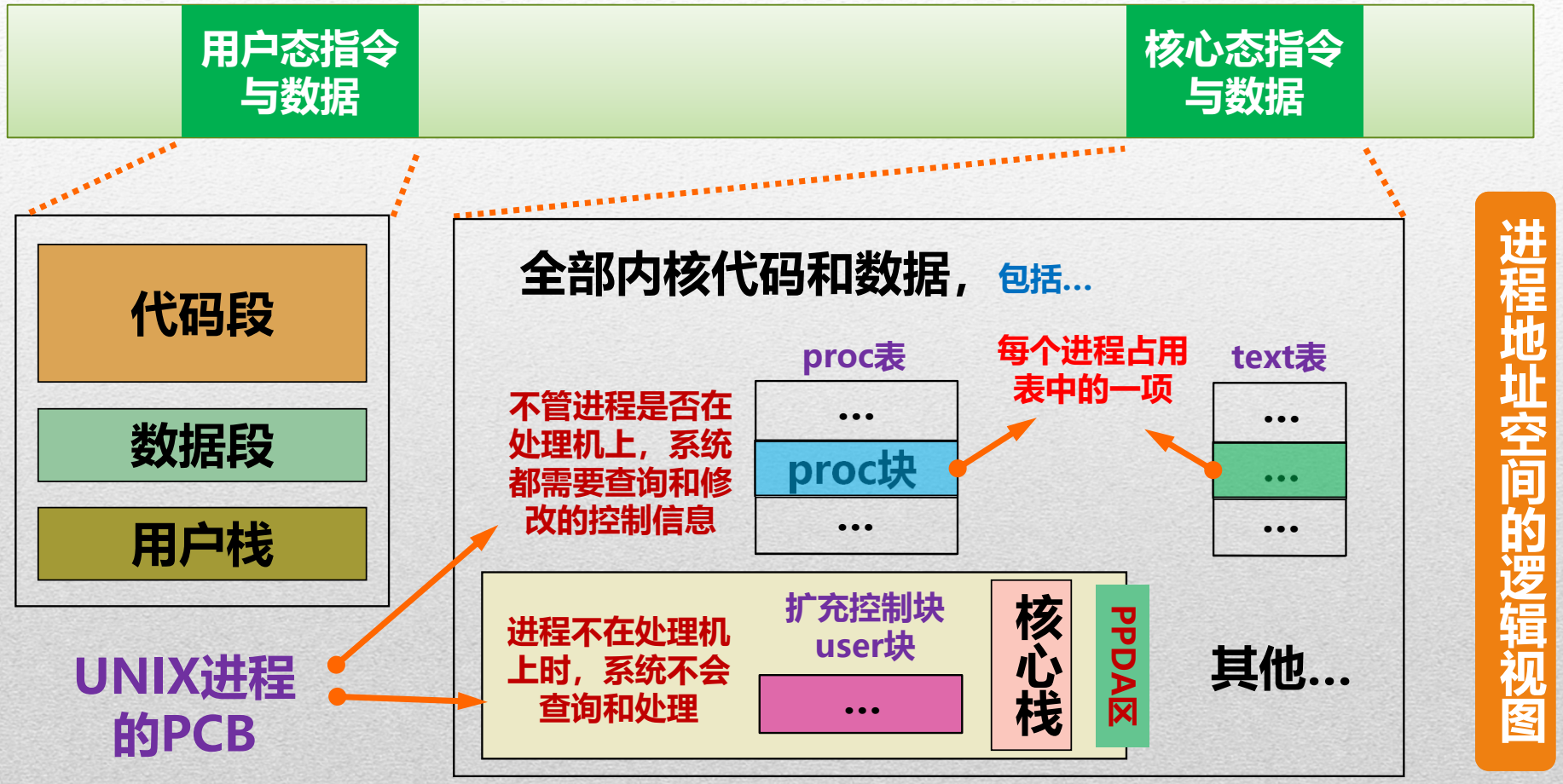
3.2 连续分配方式

3.3 页式存储管理

3.4 UNIX 存储管理

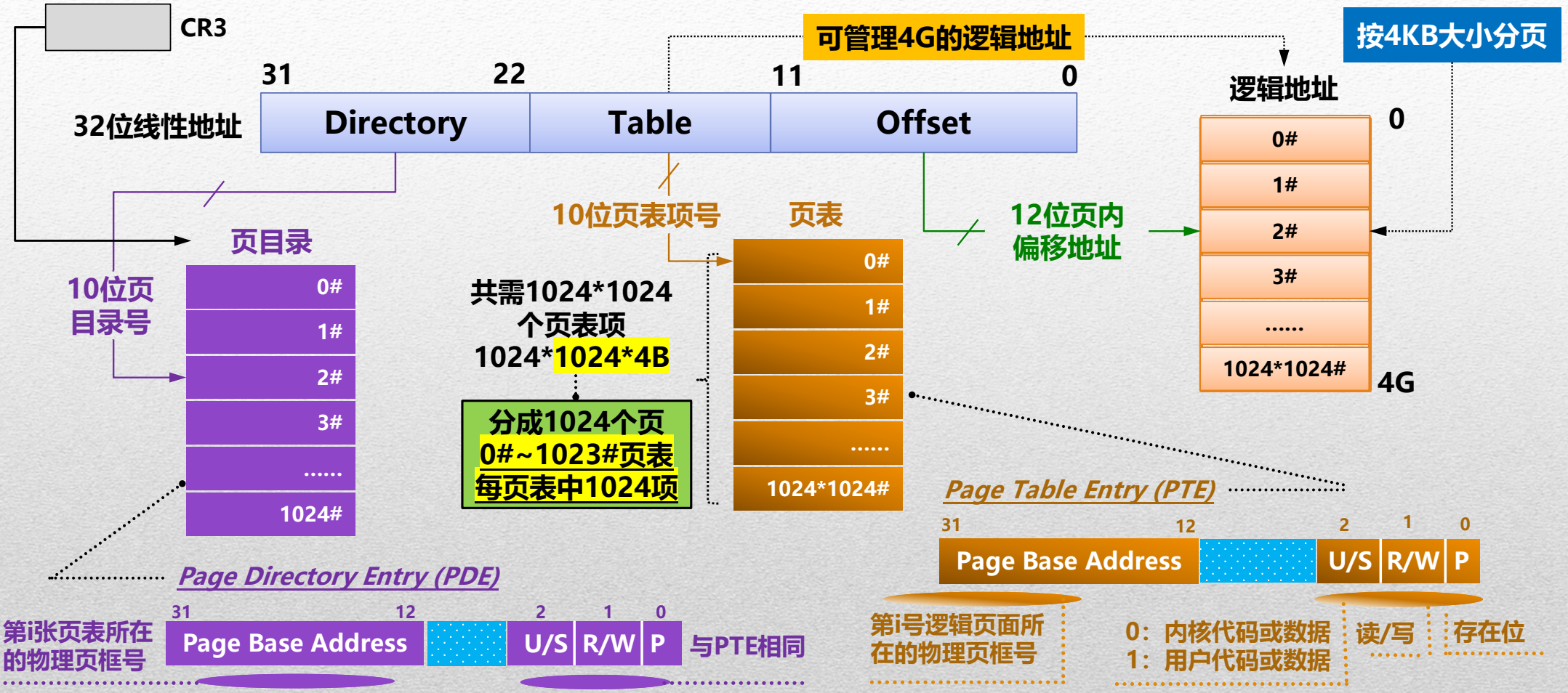
- 程序地址空间
- 物理地址空间
- 地址变换
- 存储空间管理

复习一下UNIX V6++的进程程序地址空间……

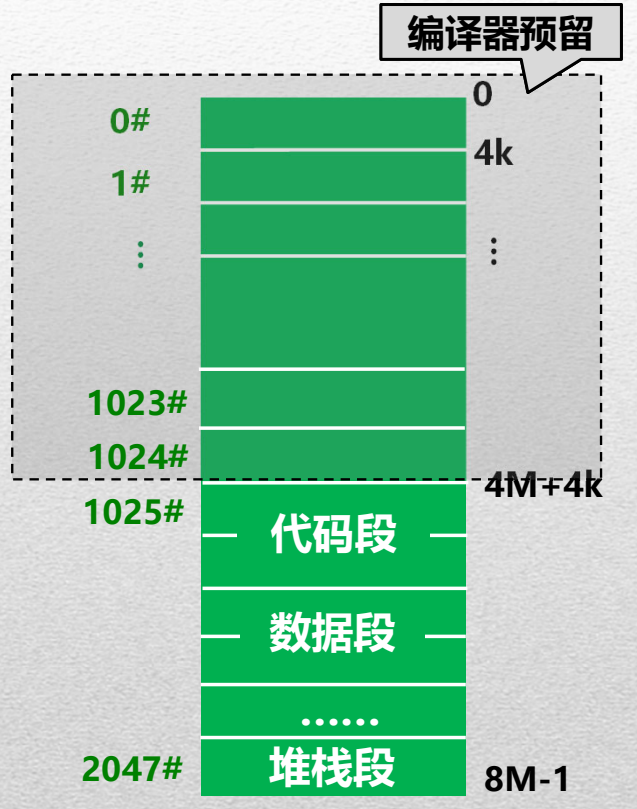


i386线性地址空间

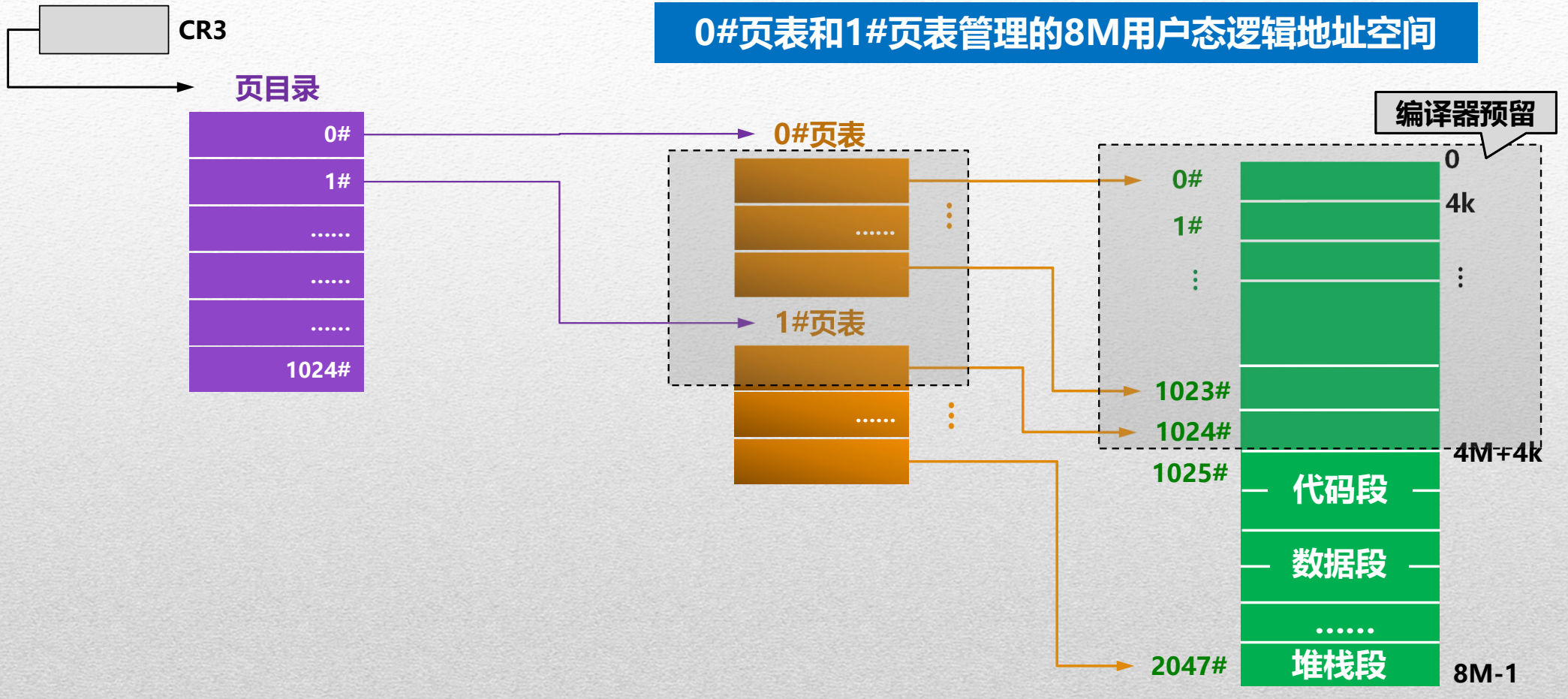
使用二级页表管理进程的线性地址



UNIX V6++的程序地址空间（线性地址、逻辑地址）

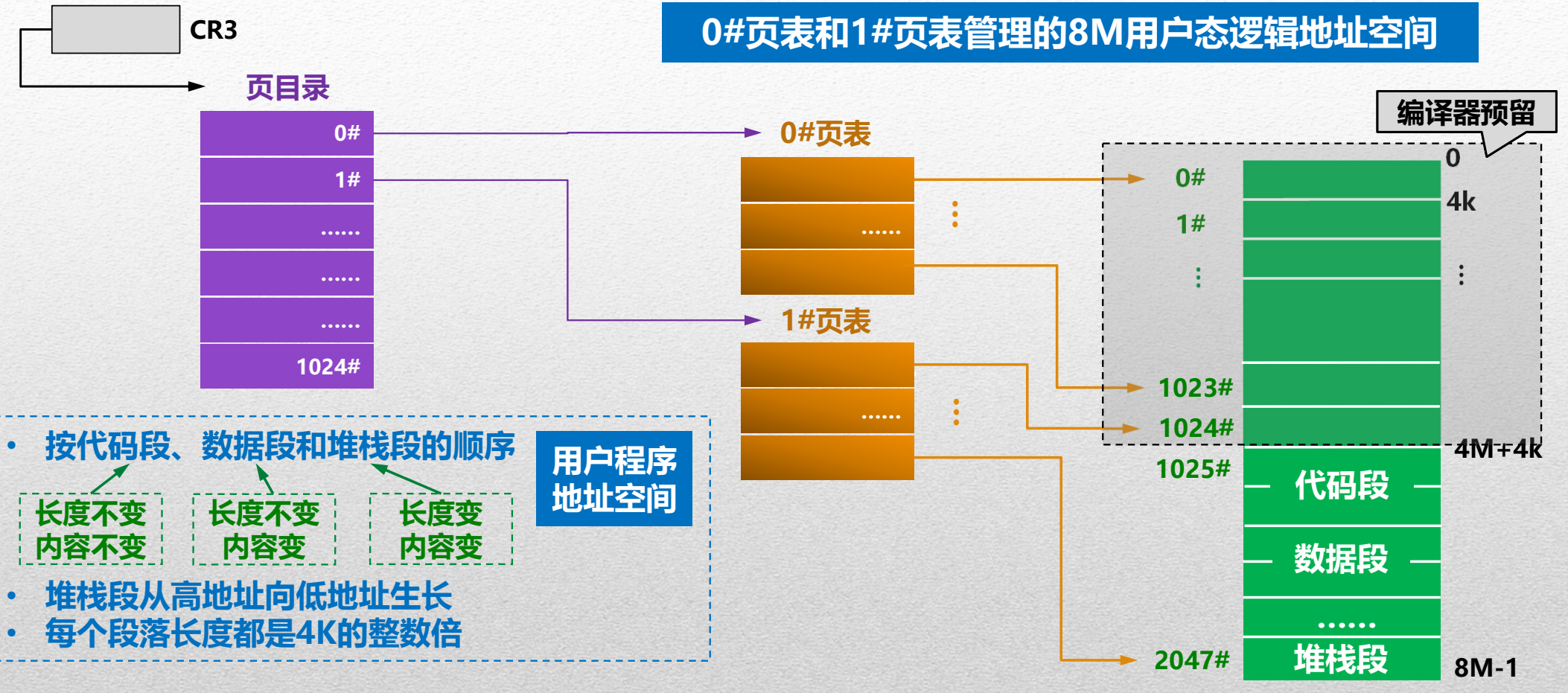


UNIX V6++的程序地址空间（线性地址、逻辑地址）

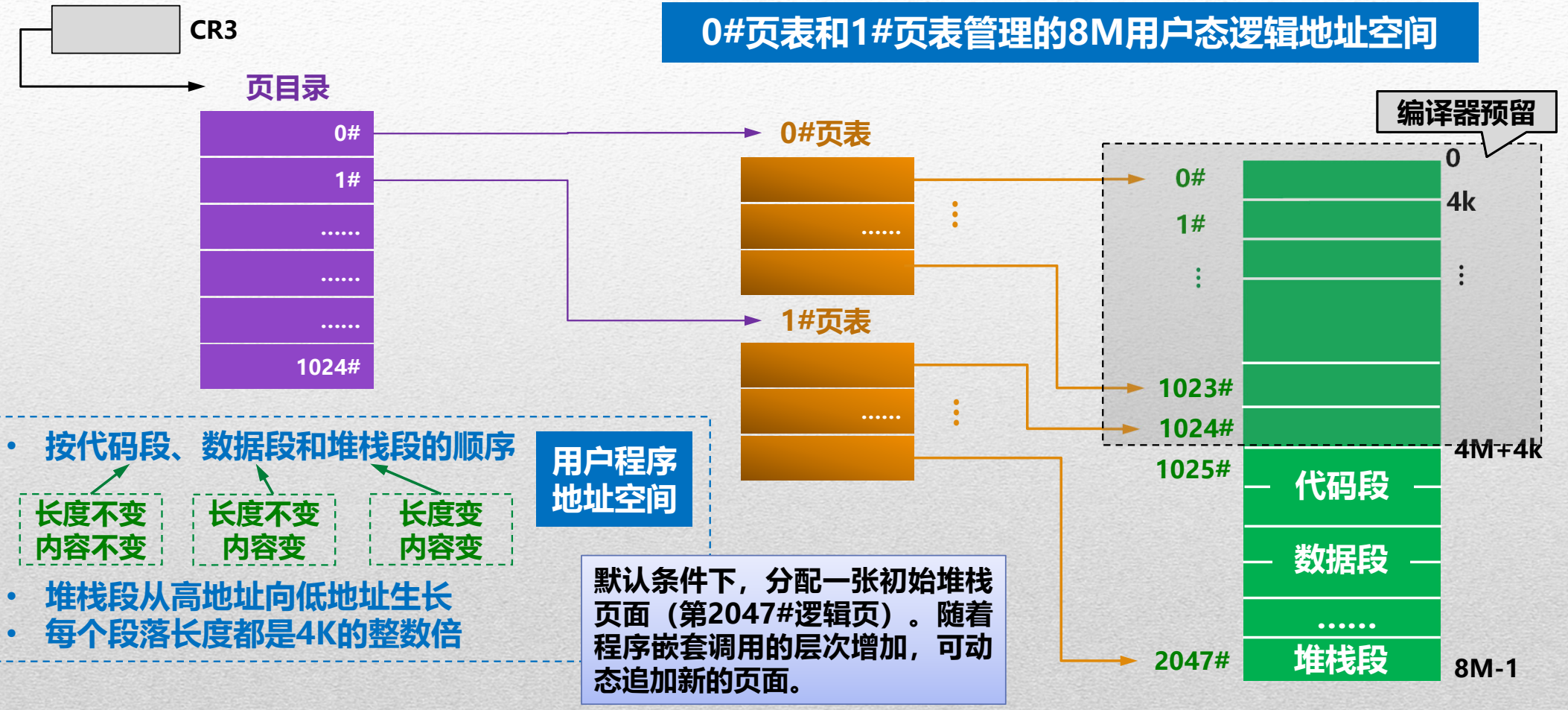


0#页表和1#页表管理的8M用户态逻辑地址空间

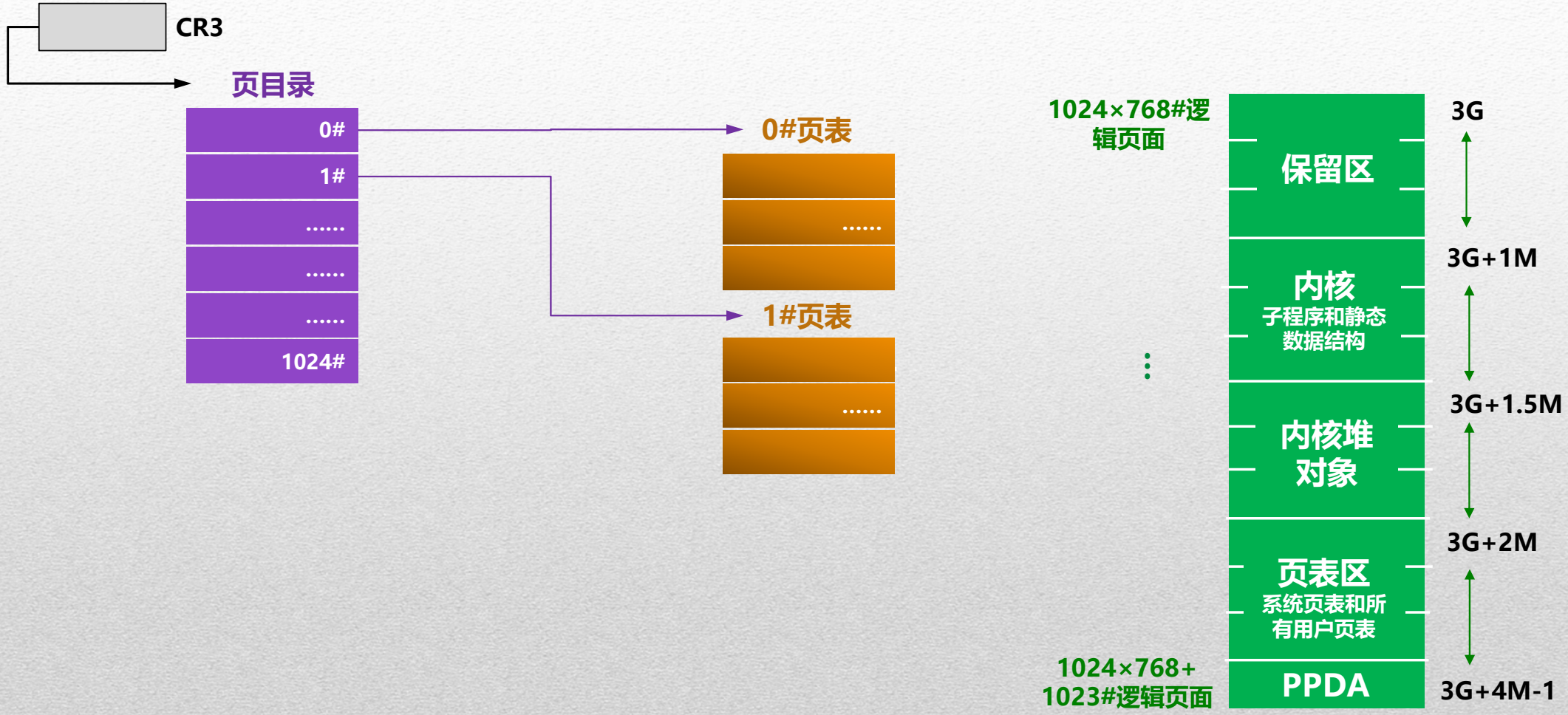
UNIX V6++的程序地址空间（线性地址、逻辑地址）



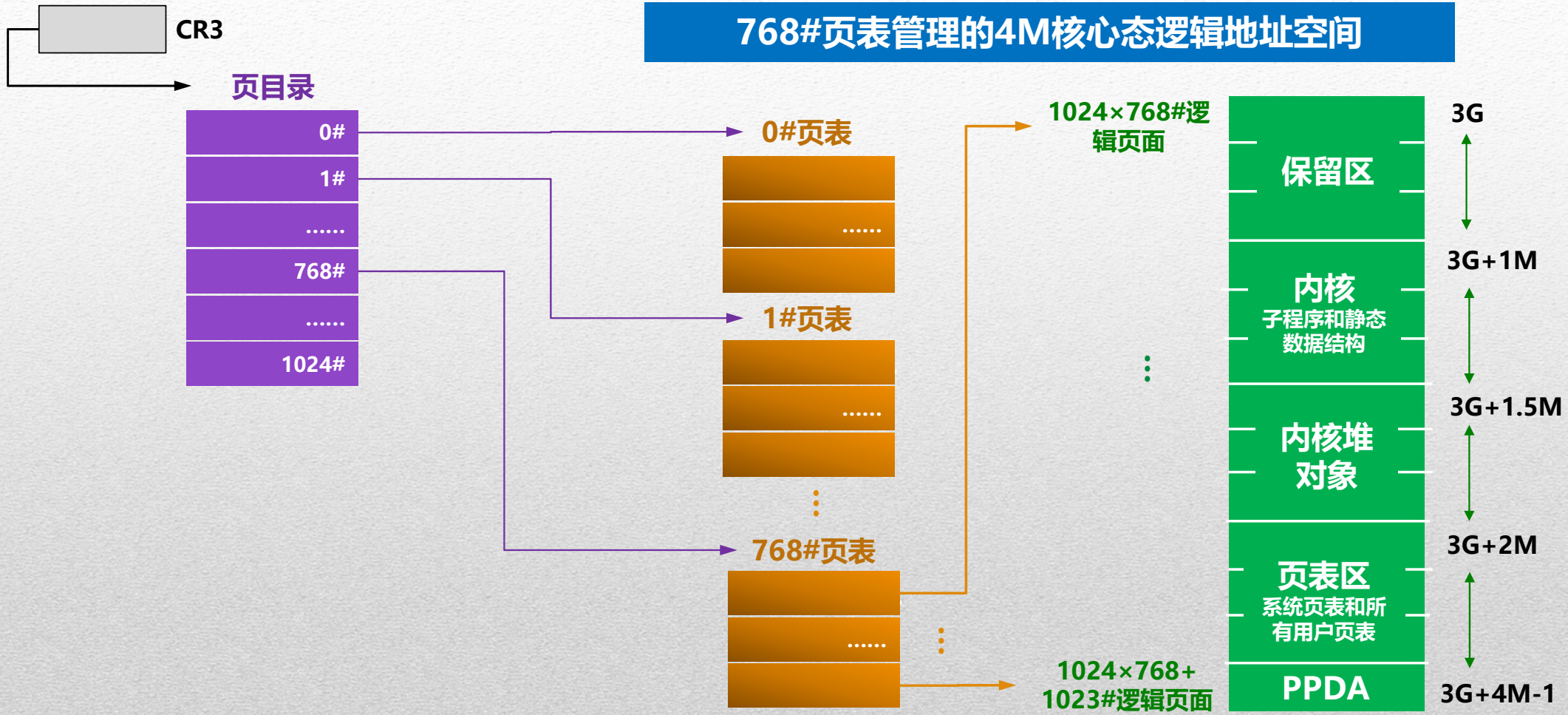
UNIX V6++的程序地址空间（线性地址、逻辑地址）



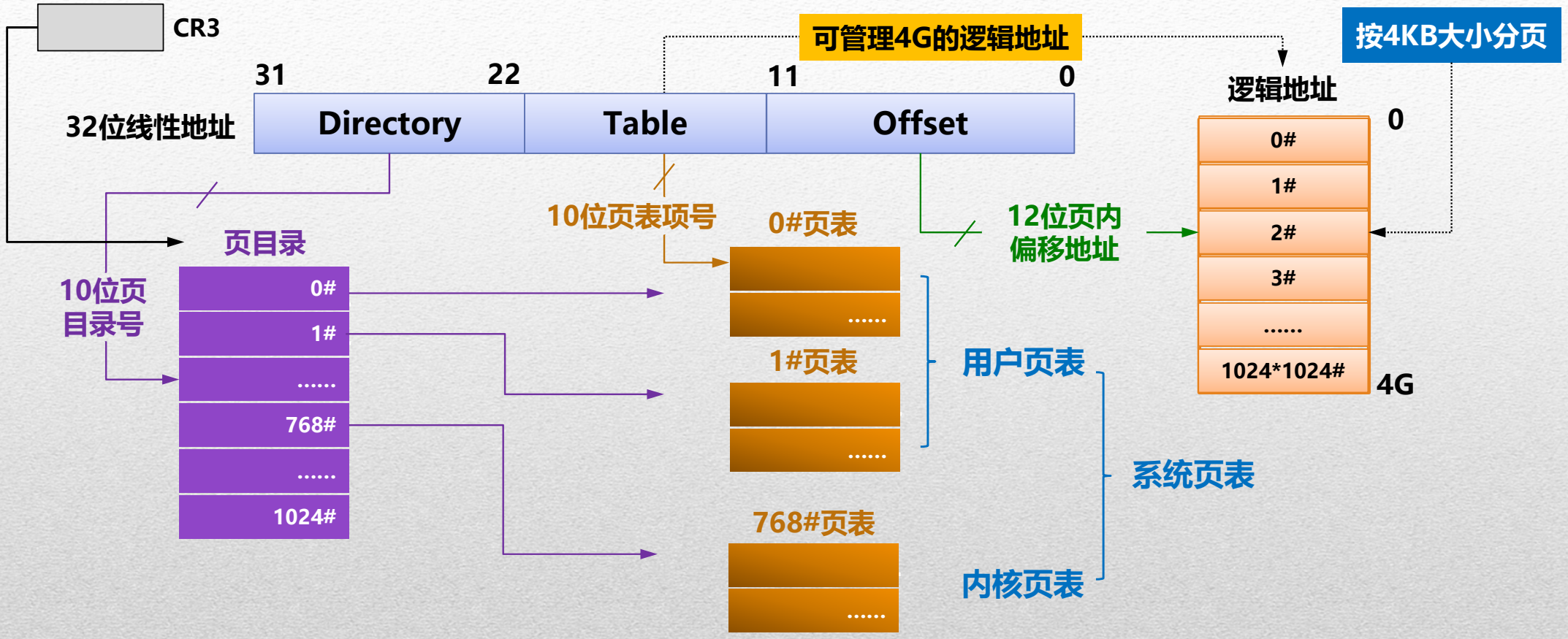
UNIX V6++的程序地址空间（线性地址、逻辑地址）



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

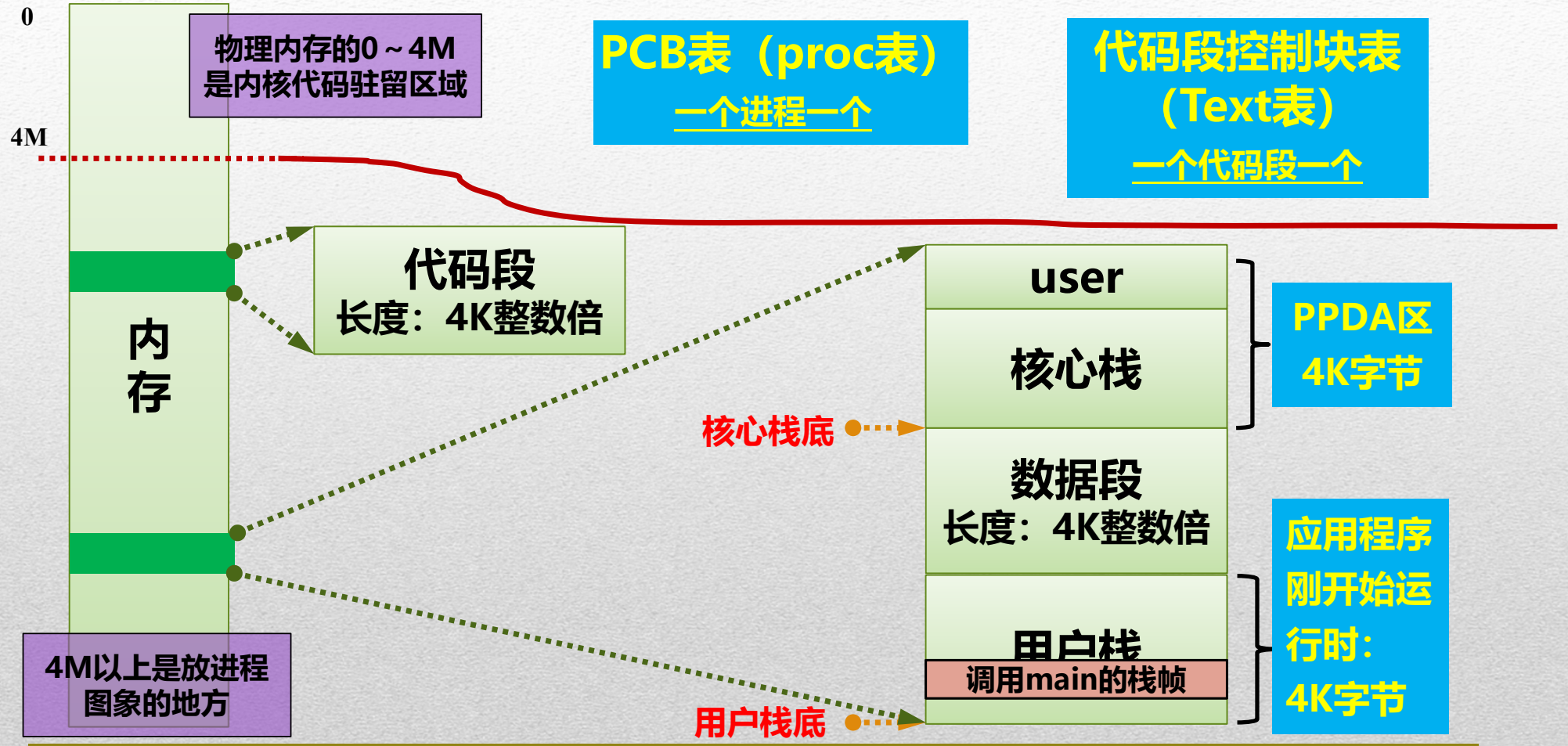
3.3 页式存储管理

3.4 UNIX 存储管理

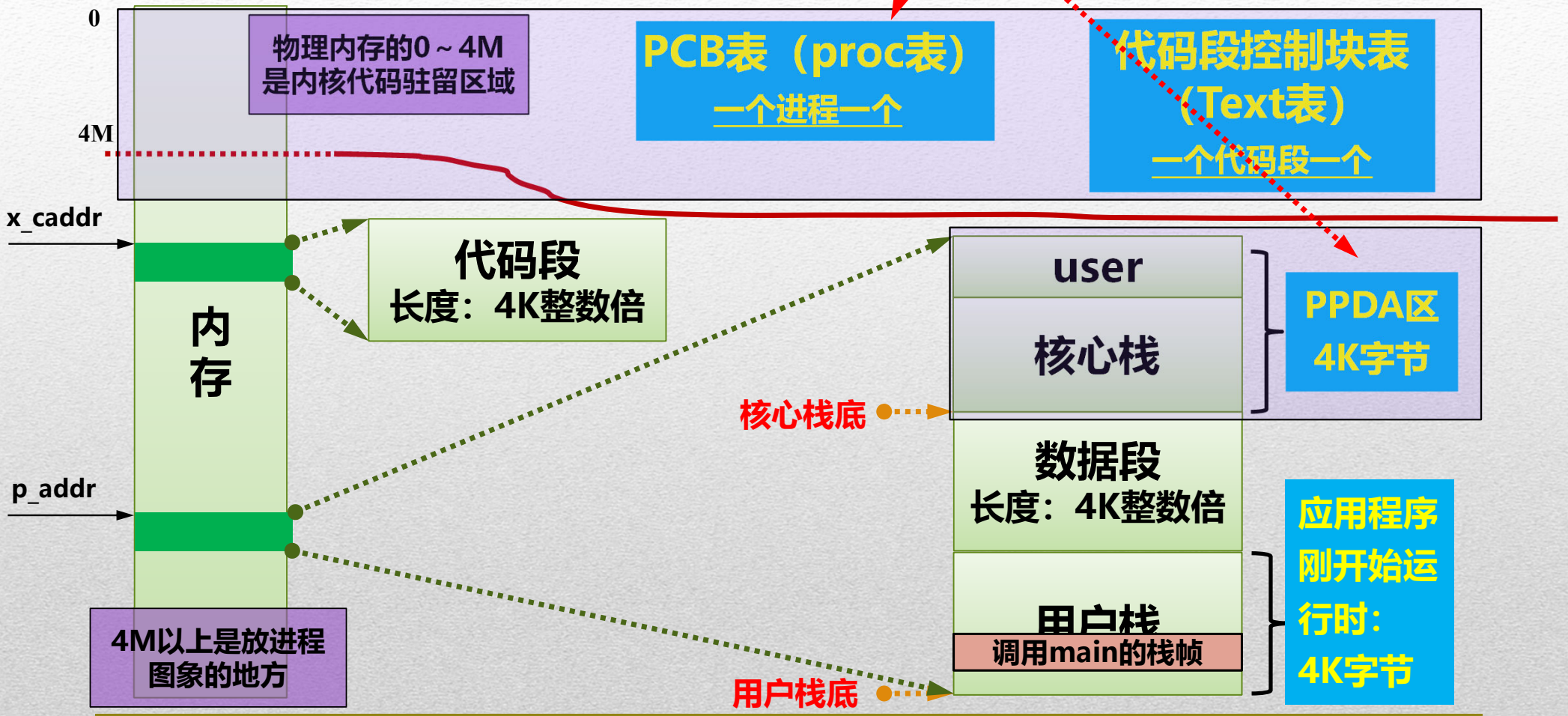
- 程序地址空间
- 物理地址空间
- 地址变换
- 存储空间管理

UNIX中进程的构成（进程图象）

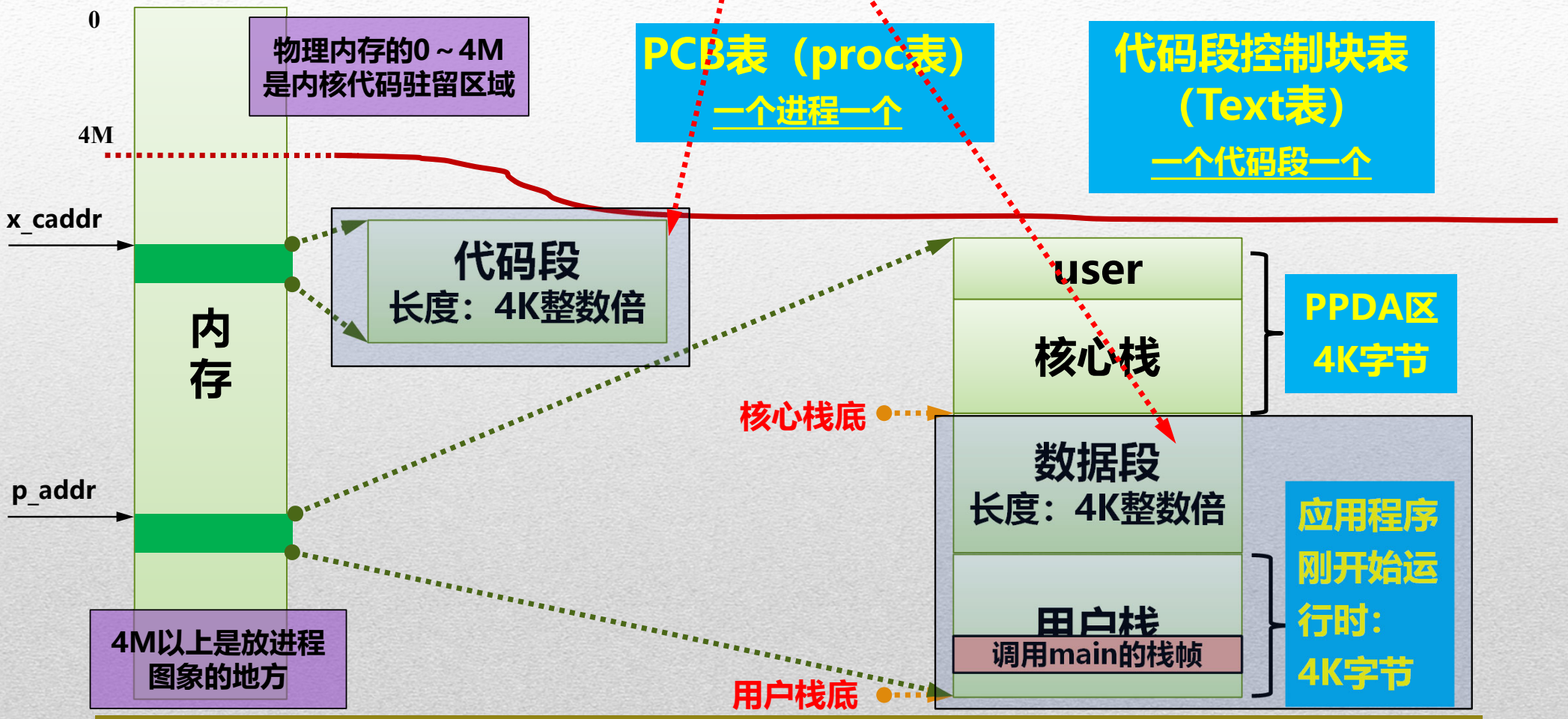
进程地址空间的物理视图



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

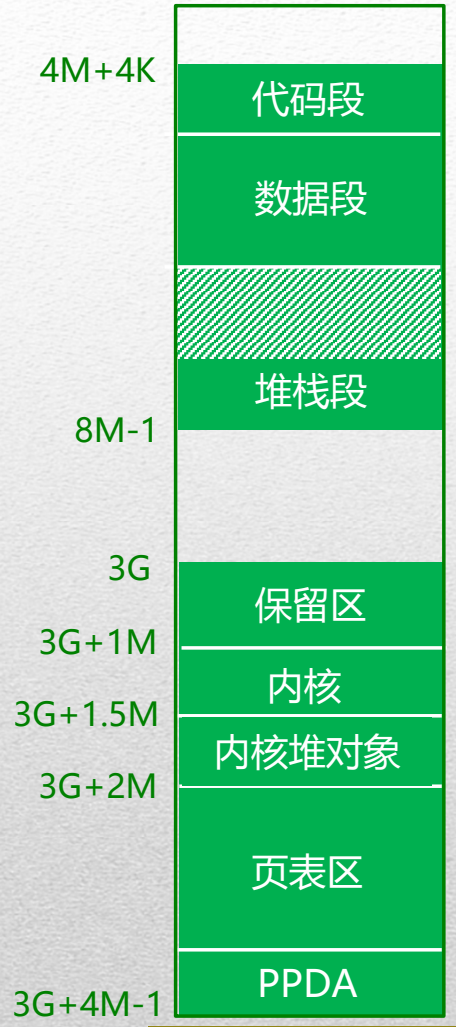
3.1 存储管理的主要任务

3.2 连续分配方式

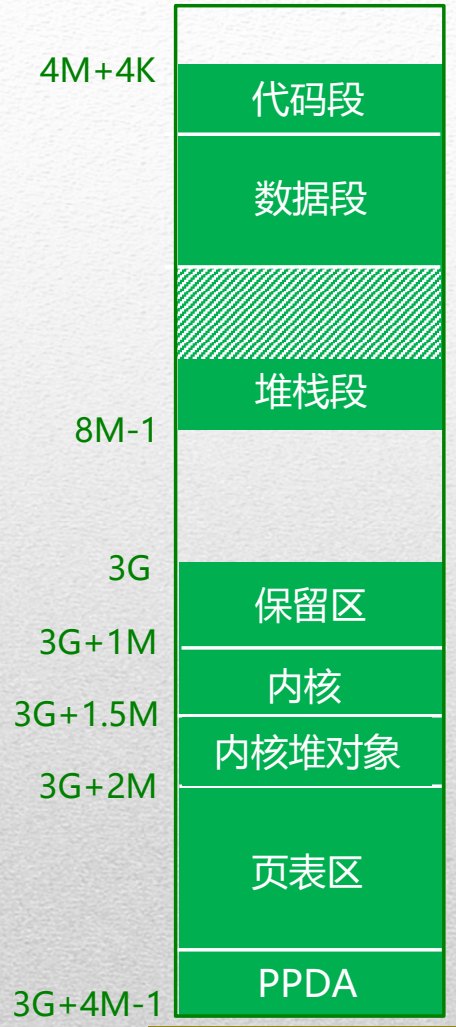
3.3 页式存储管理

3.4 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

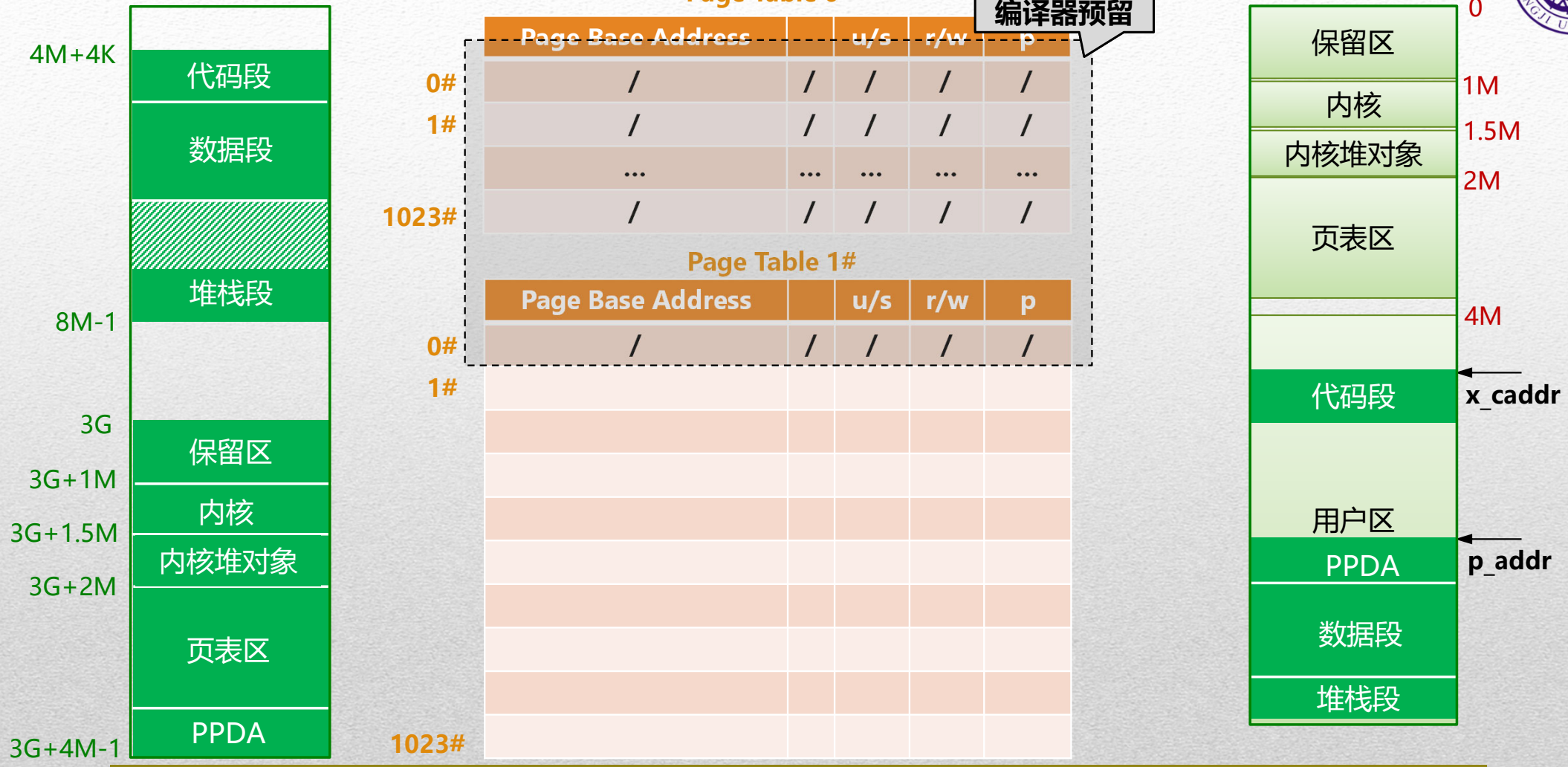


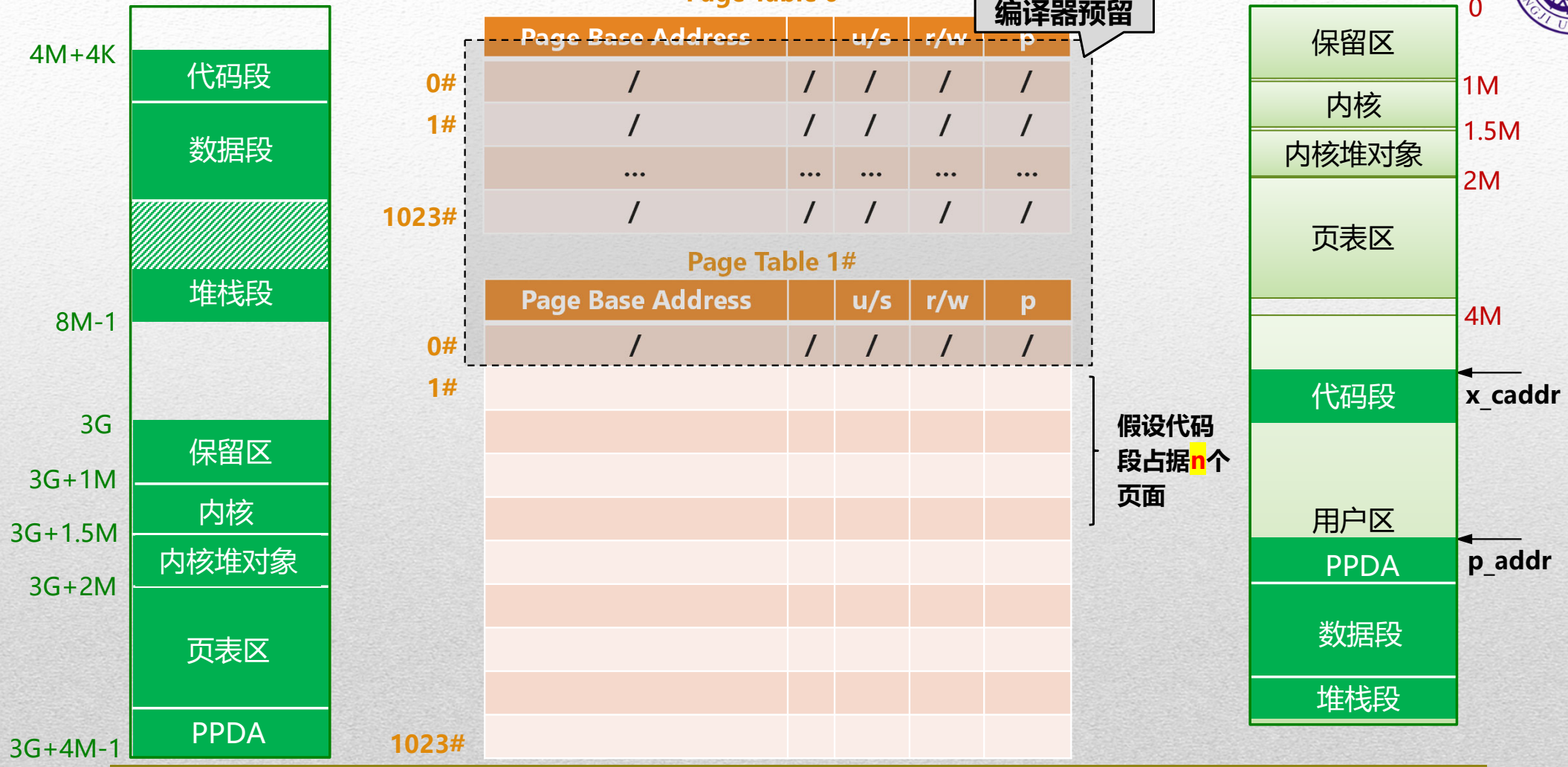
Page Table 0#

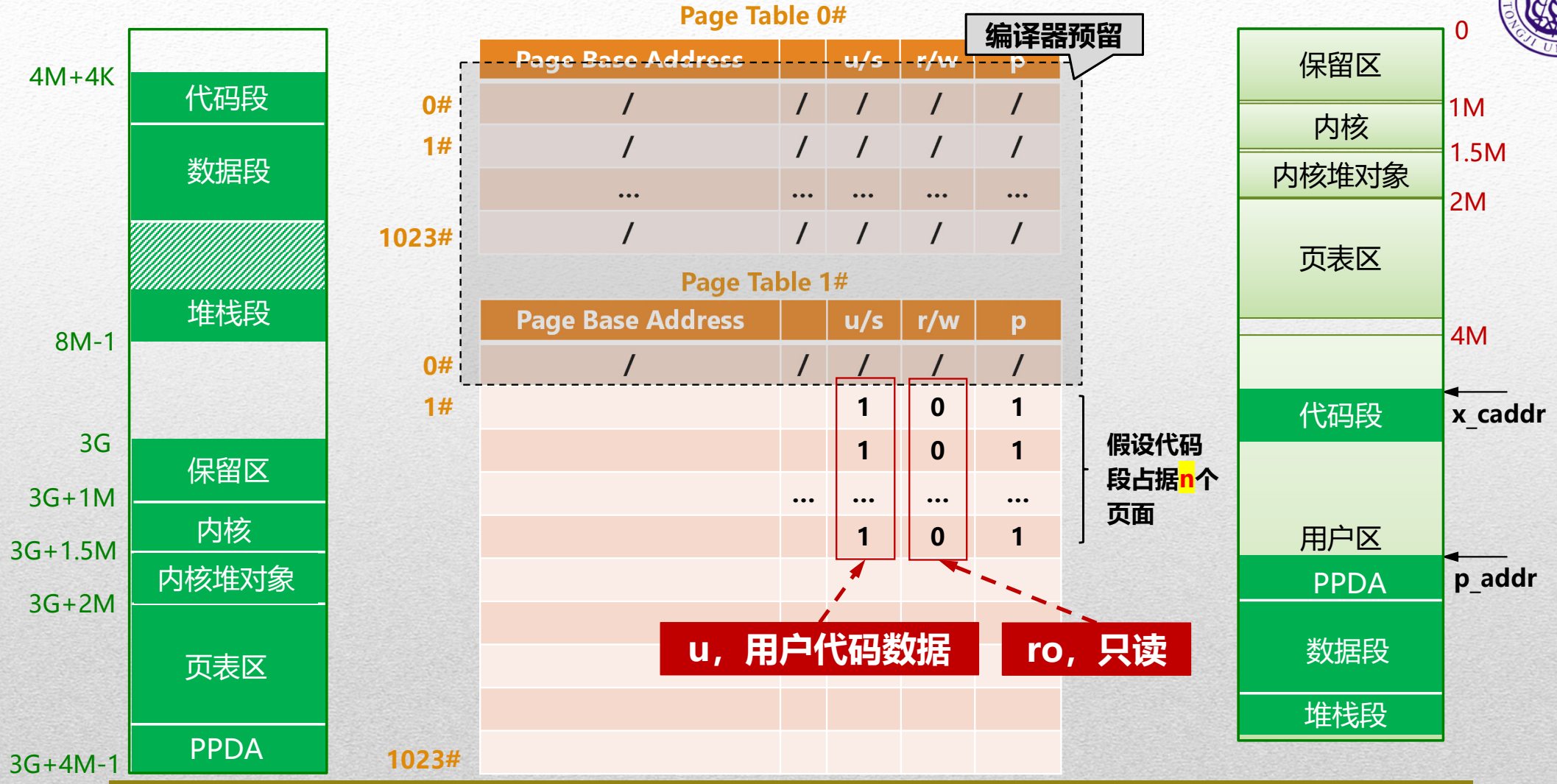
	Page Base Address		u/s	r/w	p
0#					
1#					
1023#					

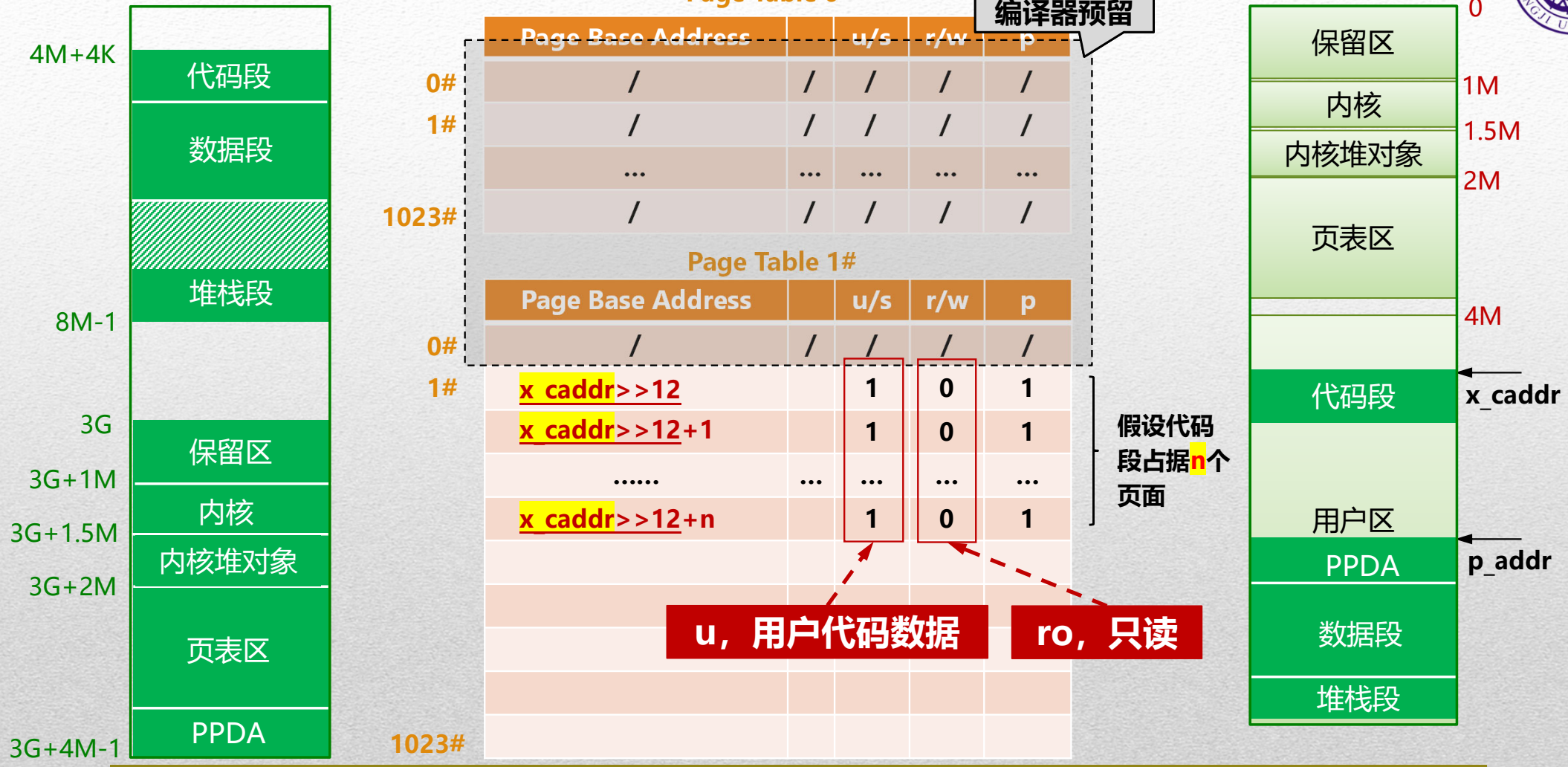
Page Table 1#

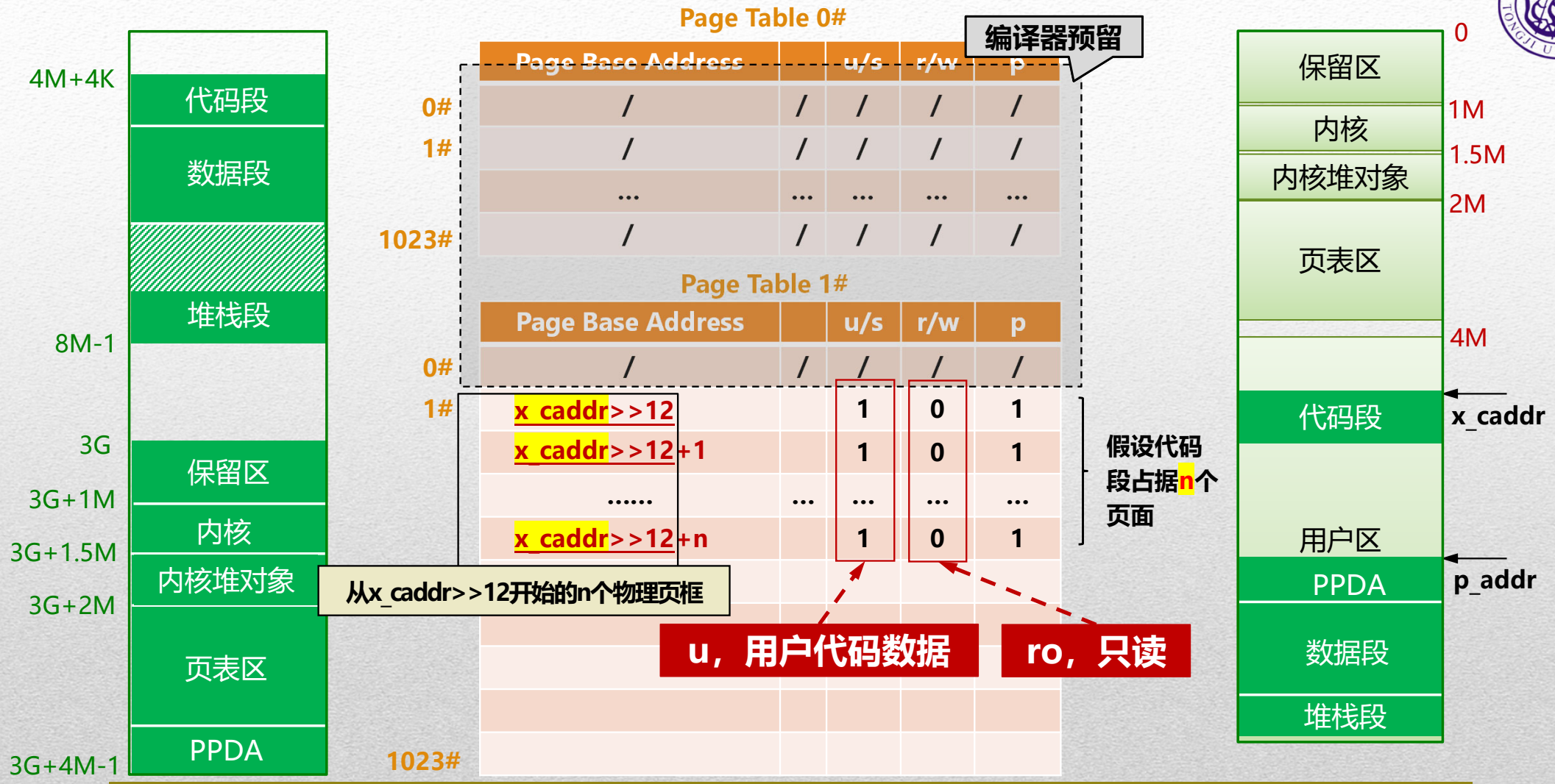
	Page Base Address		u/s	r/w	p
0#					
1#					
1023#					

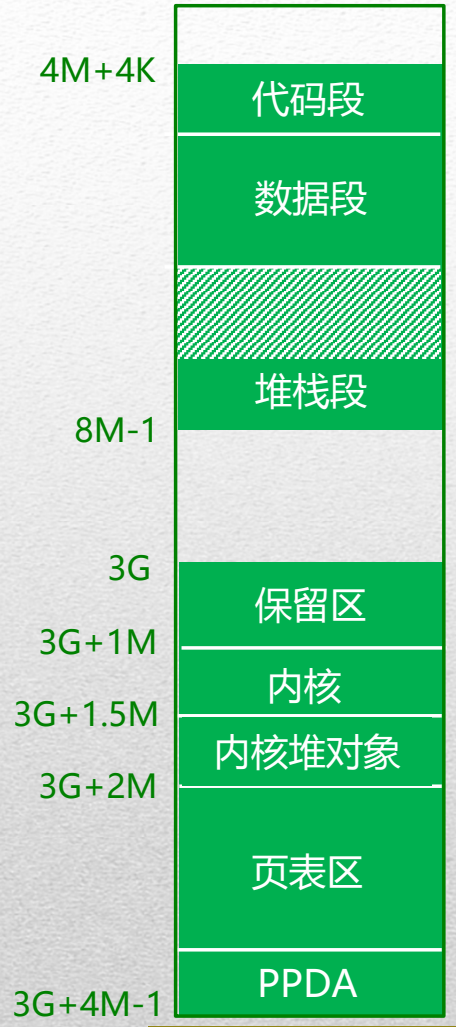

Page Table 0#
编译器预留
Page Base Address
u/s
r/w
p
0#
1#
1023#
Page Table 1#
4M+4K
代码段
数据段
堆栈段
8M-1
3G
保留区
3G+1M
内核
3G+1.5M
内核堆对象
3G+2M
页表区
PPDA
3G+4M-1
0
保留区
1M
内核
1.5M
内核堆对象
2M
页表区
4M
代码段
x_caddr
用户区
PPDA
p_addr
数据段
堆栈段


Page Table 0#
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
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#
假设代码段占据n个页面
u，用户代码数据
ro，只读
4M+4K
8M-1
3G
3G+1M
3G+1.5M
3G+2M
3G+4M-1
代码段
数据段
堆栈段
保留区
内核
内核堆对象
页表区
PPDA
0
1M
1.5M
2M
4M
x_caddr
p_addr
用户区


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个页面
u，用户代码数据
ro，只读
4M+4K
8M-1
3G
3G+1M
3G+1.5M
3G+2M
3G+4M-1
代码段
数据段
堆栈段
保留区
内核
内核堆对象
页表区
PPDA
0
1M
1.5M
2M
4M
x_caddr
p_addr
用户区


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个页面
从x_caddr>>12开始的n个物理页框
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#	/	/	/	/	/

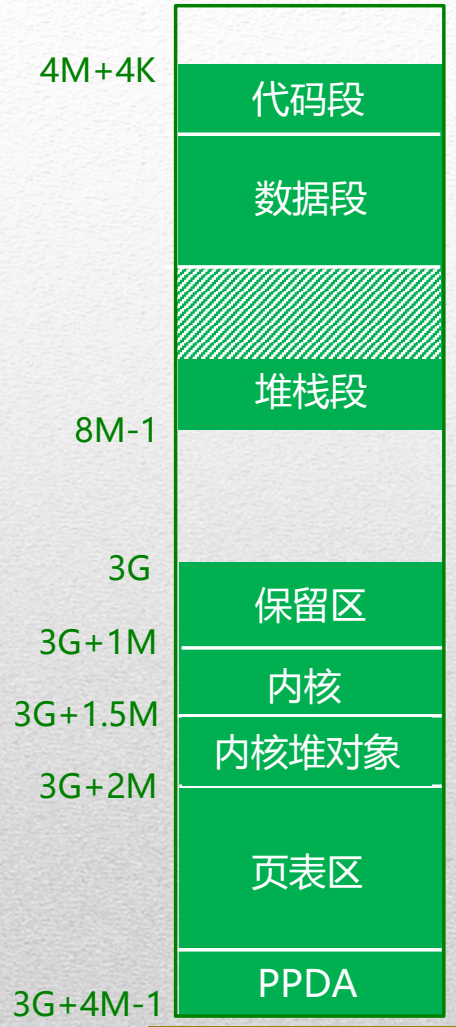
编译器预留

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个页面

假设数据段占据m个页面



Page Table 0#

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

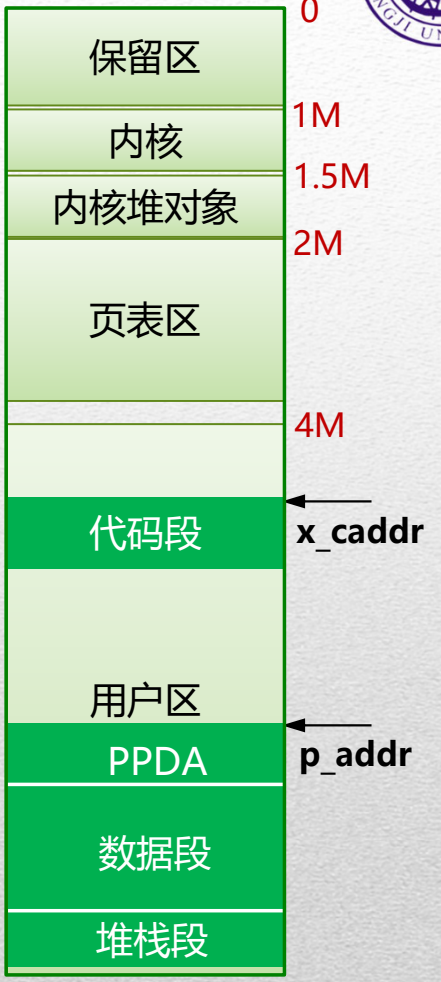
编译器预留

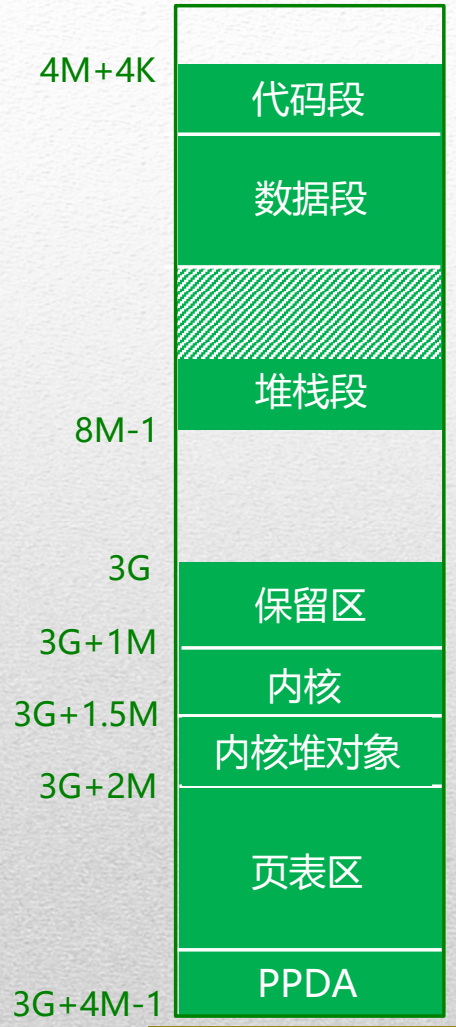
Page Table 1#

	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个页面

假设数据段占据m个页面





Page Table 0#

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

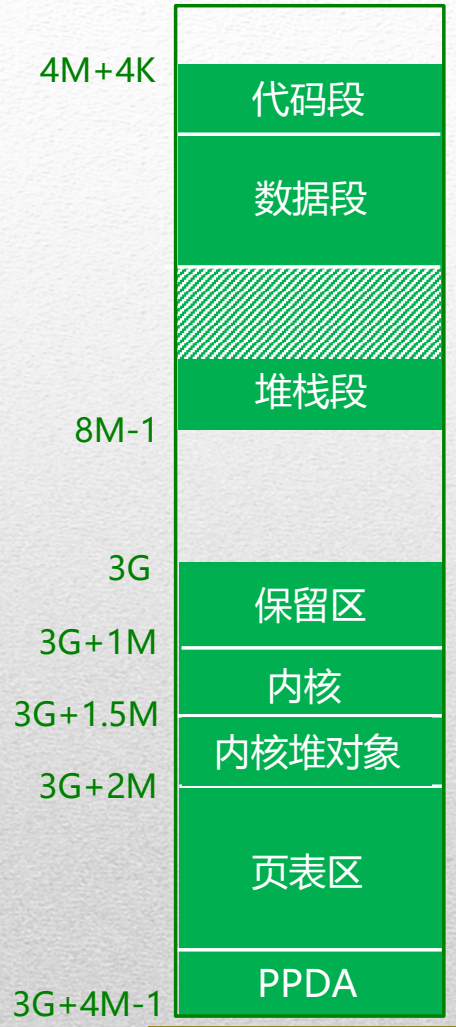
编译器预留

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个页面

假设数据段占据m个页面



Page Table 0#

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

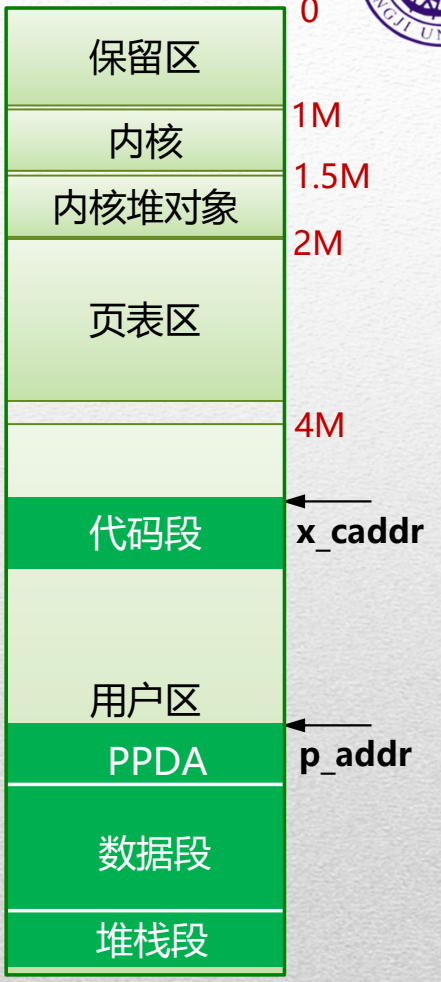
编译器预留

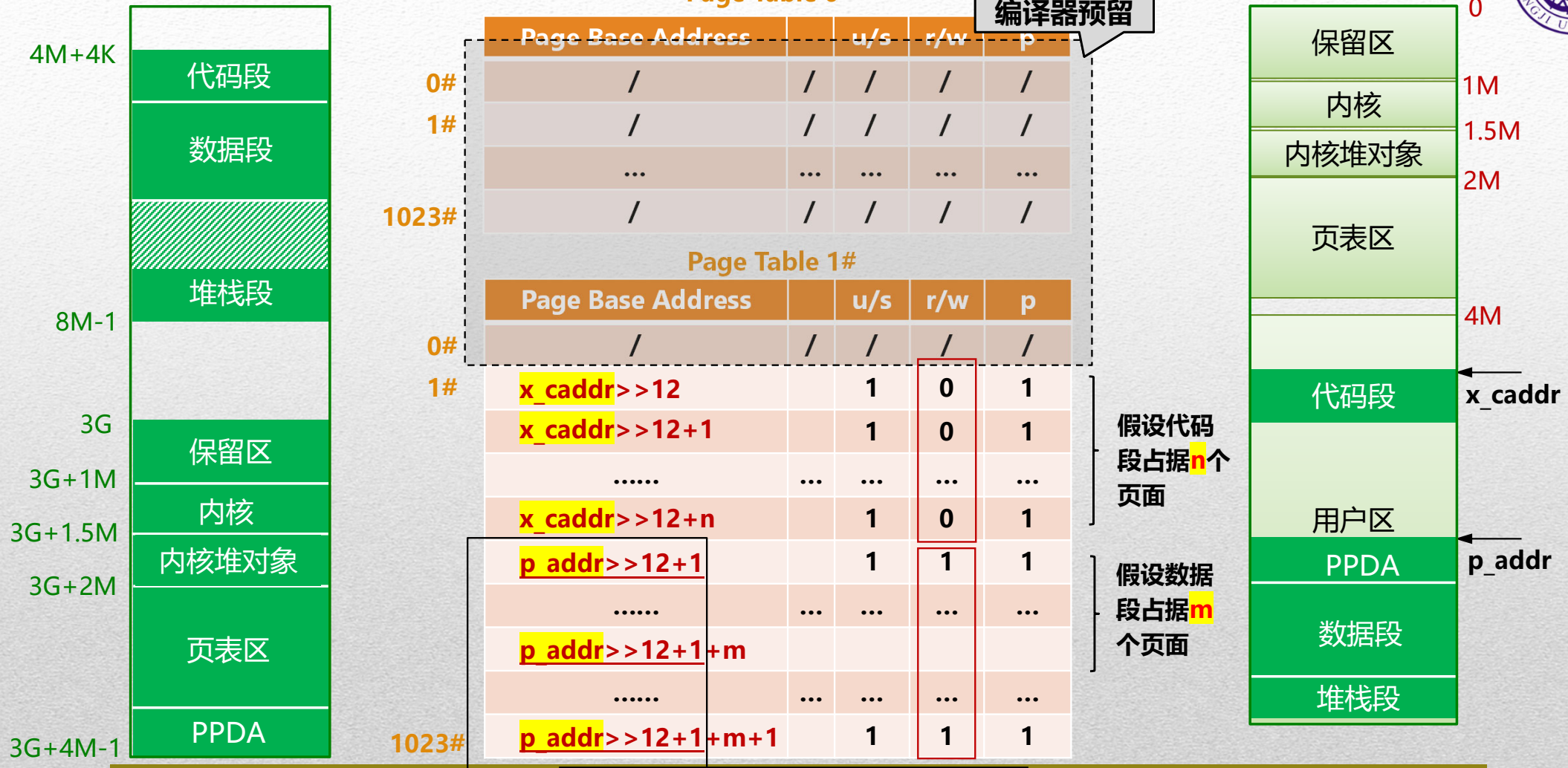
Page Table 1#

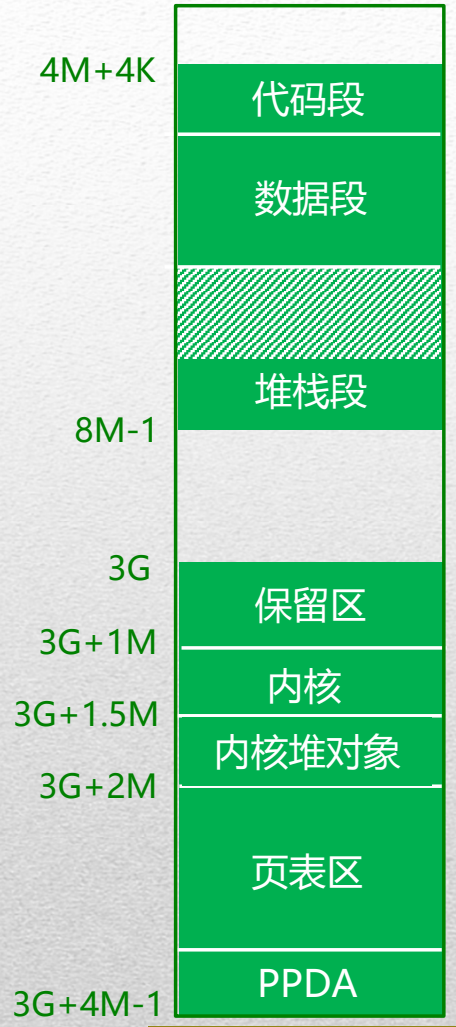
	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
	p_addr>>12+1		1	1	1
	.....	...	...	...	...
	p_addr>>12+1+m				
	.....	...	...	...	...
1023#	p_addr>>12+1+m+1		1	1	1

假设代码段占据n个页面

假设数据段占据m个页面

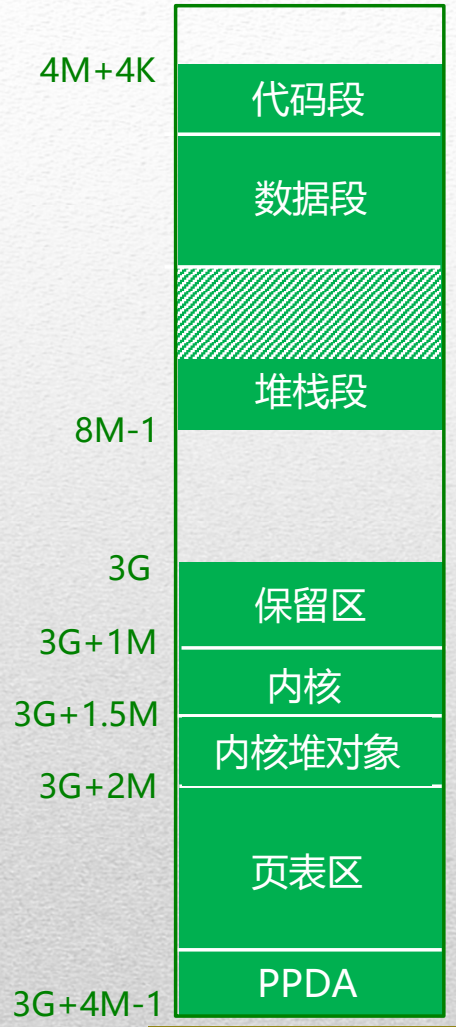



Page Table 0#
编译器预留
Page Base Address | | u/s | r/w | p
0# / / / / /
1# / / / / /
... ...
1023# / / / / /
Page Table 1#
Page Base Address | | u/s | r/w | p
0# / / / / /
1# x_caddr>>12 1 0 1
x_caddr>>12+1 1 0 1
....
x_caddr>>12+n 1 0 1
p_addr>>12+1 1 1 1
....
p_addr>>12+1+m
....
1023# p_addr>>12+1+m+1 1 1 1
假设代码段占据n个页面
假设数据段占据m个页面
从p_addr>>12+1开始的m+1个物理页框
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 768#

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

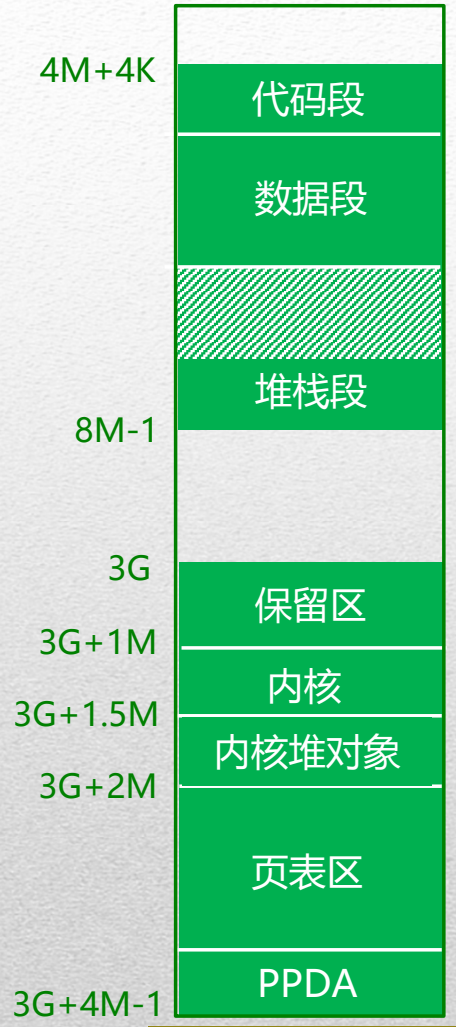


Page Table 768#

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

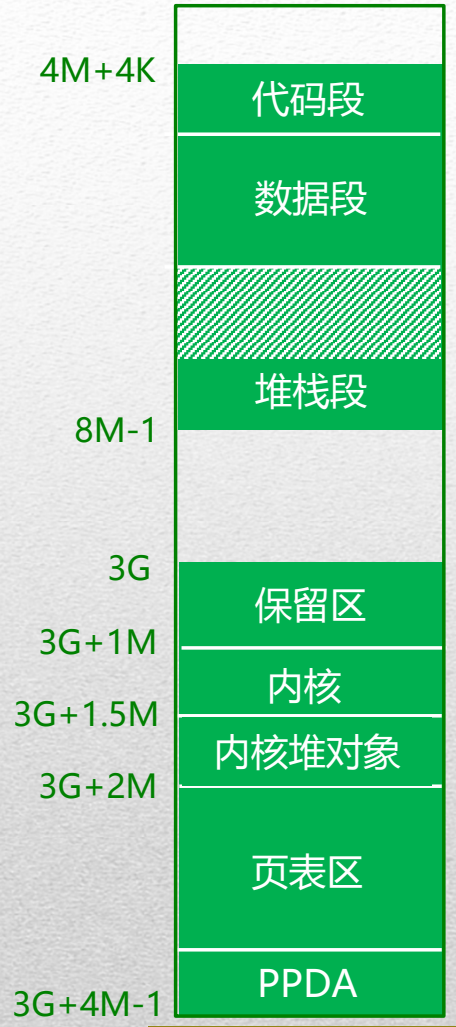
s，内核代码数据

rw，读写



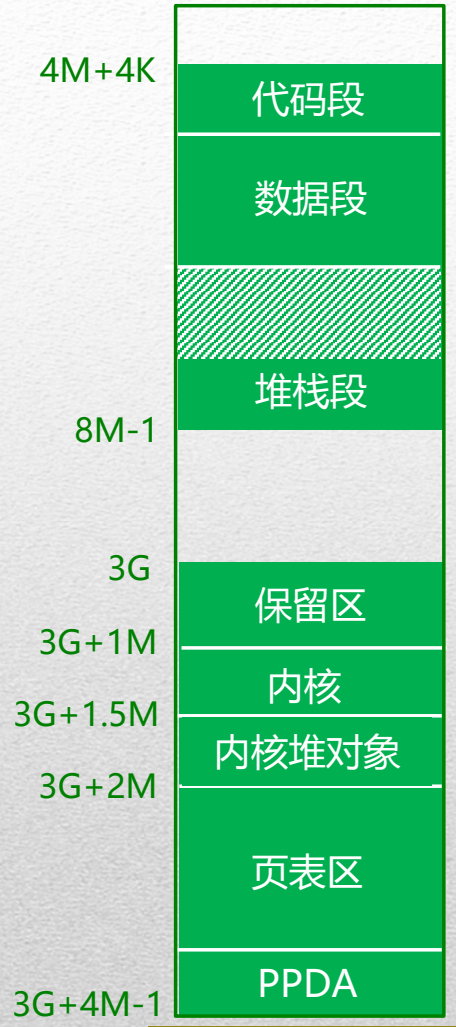
Page Table 768#

	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
/	/	/	/	/

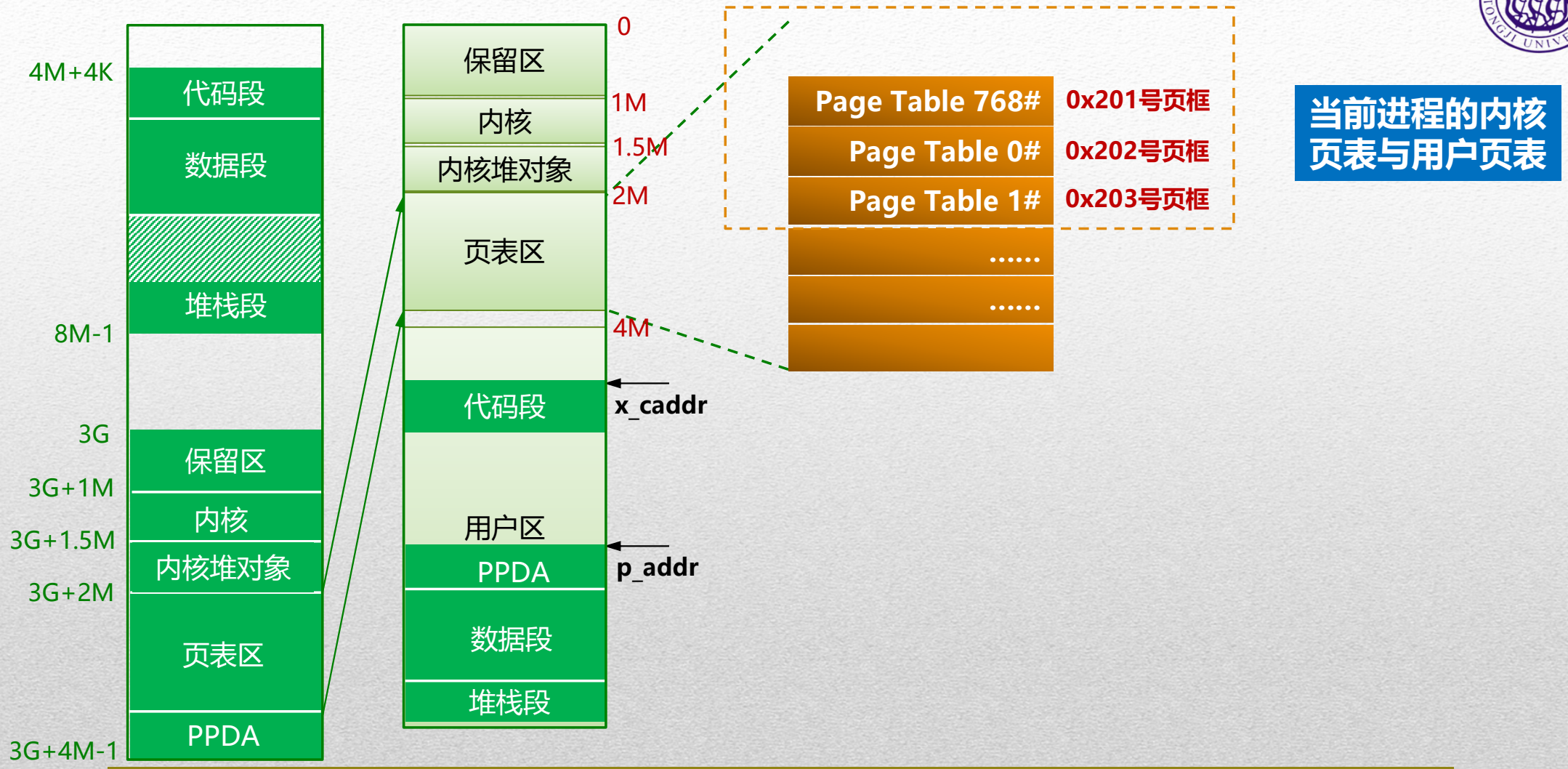
Page Table 1#

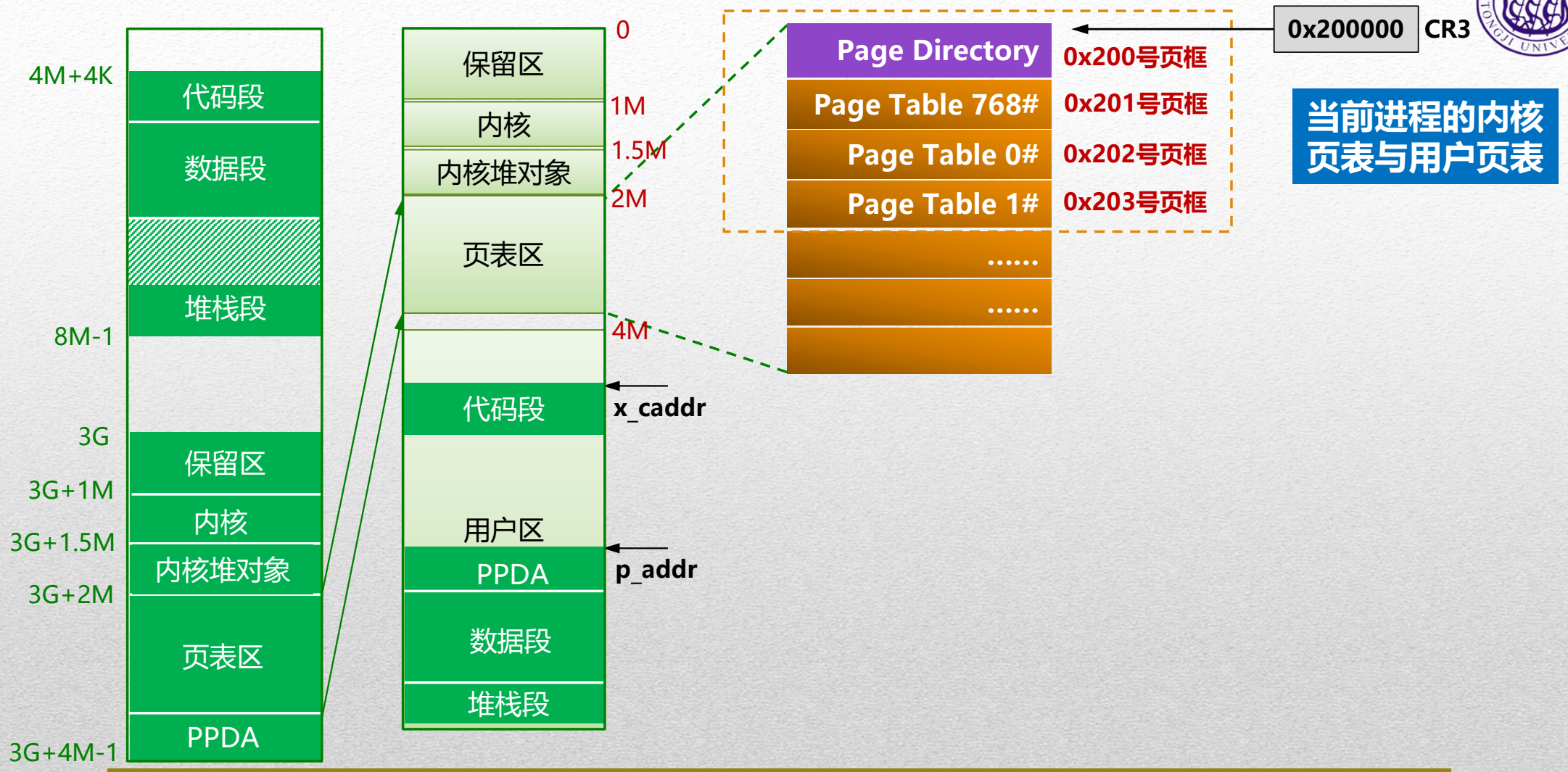
	Page Base Address		u/s	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	.....	...	...	...	...
	x_caddr>>12+n		1	0	1
	p_addr>>12+1		1	1	1
	.....	...	...	...	...
	p_addr>>12+1+m				
	.....	...	...	...	...
1023#	p_addr>>12+1+m+1		1	1	1

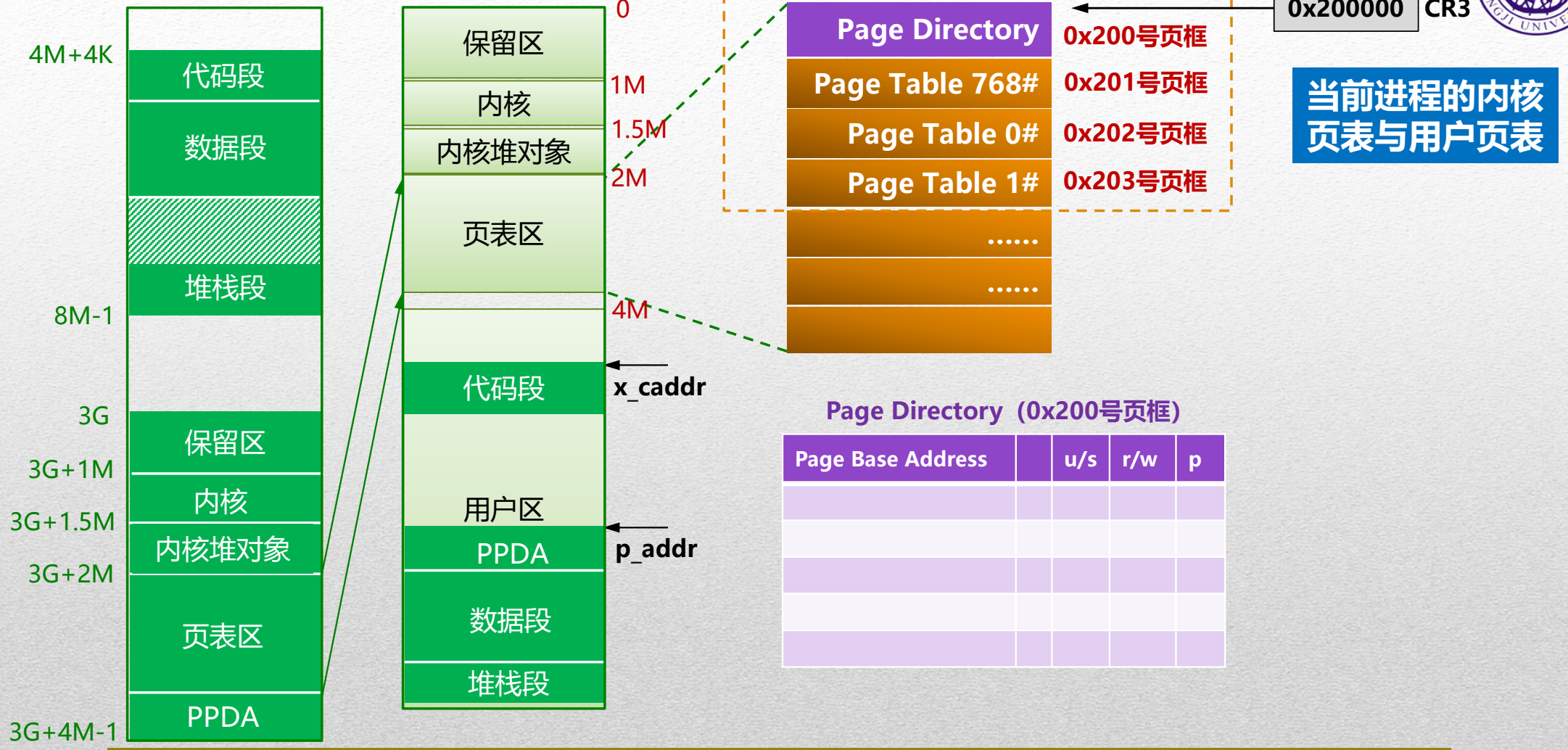
Page Table 768#

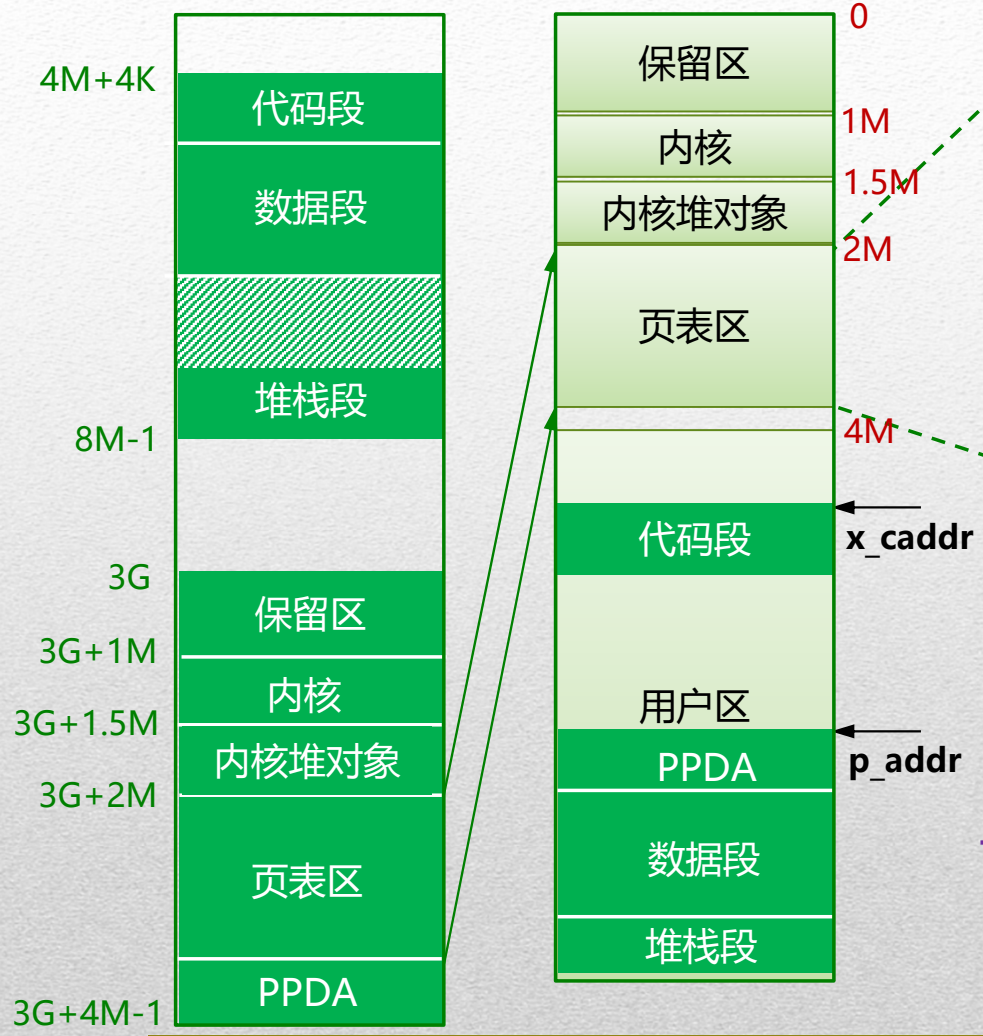
	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....	...	...	...	...
1023#	p_addr>>12		0	1	1

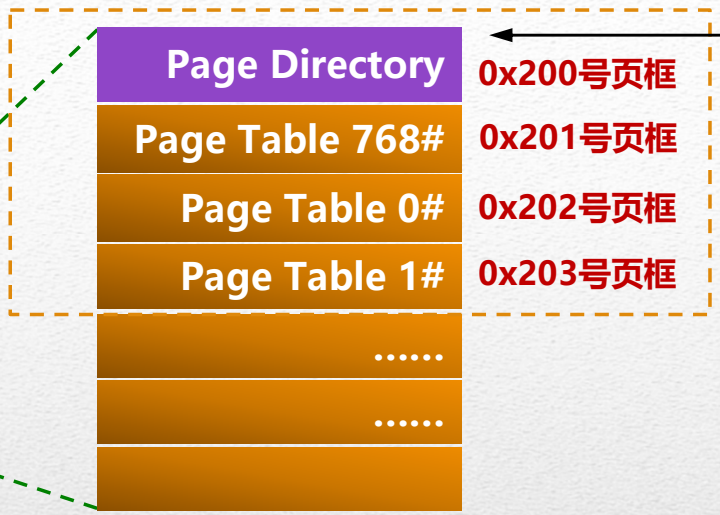
三张页表保存在哪里?


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 768#
0x201号页框
Page Table 0#
0x202号页框
Page Table 1#
0x203号页框
.....
.....
当前进程的内核
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
页表区
3G+4M-1
PPDA
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 Directory
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
.....
.....
0x200000
CR3
当前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
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 Directory
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
.....
.....
0x200000
CR3
当前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

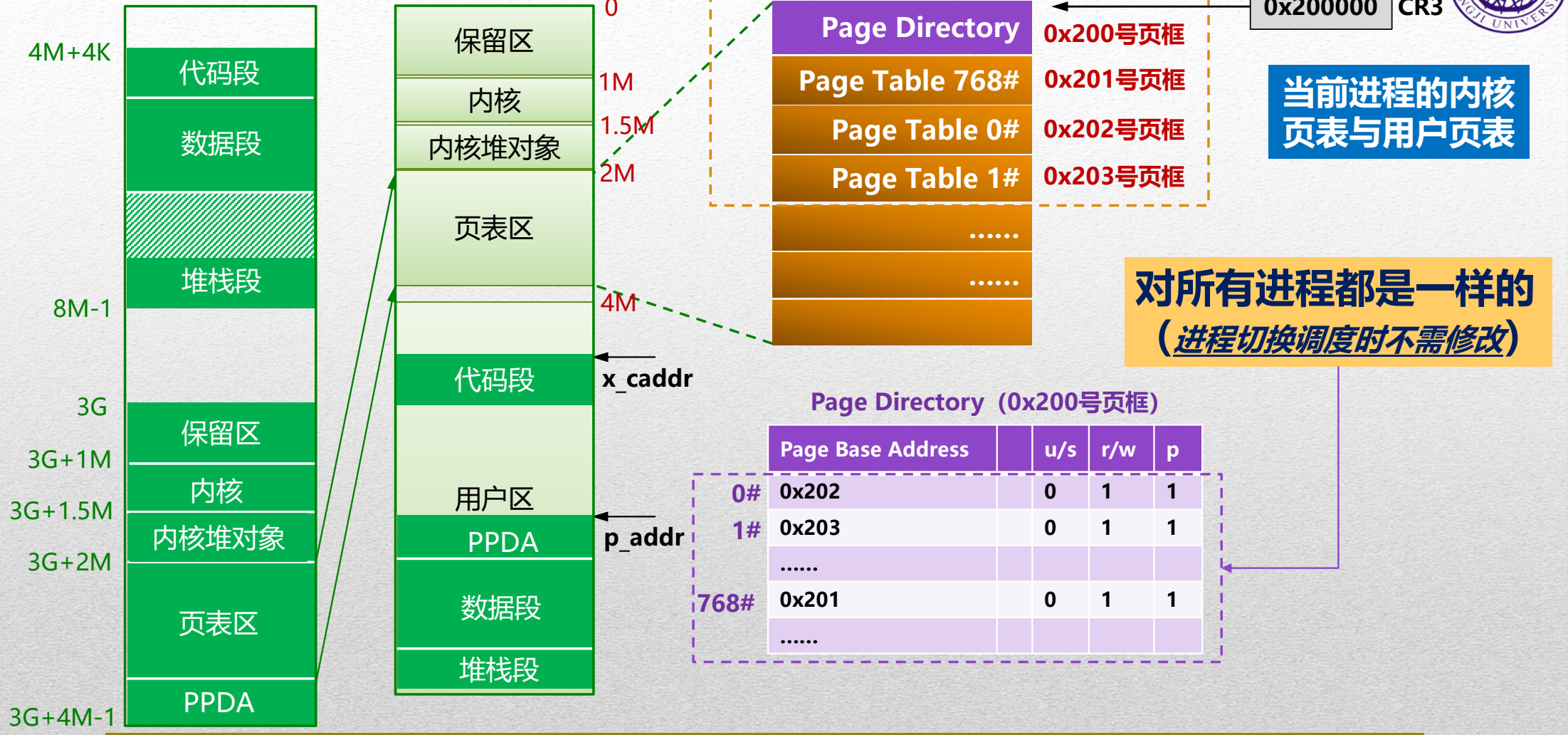


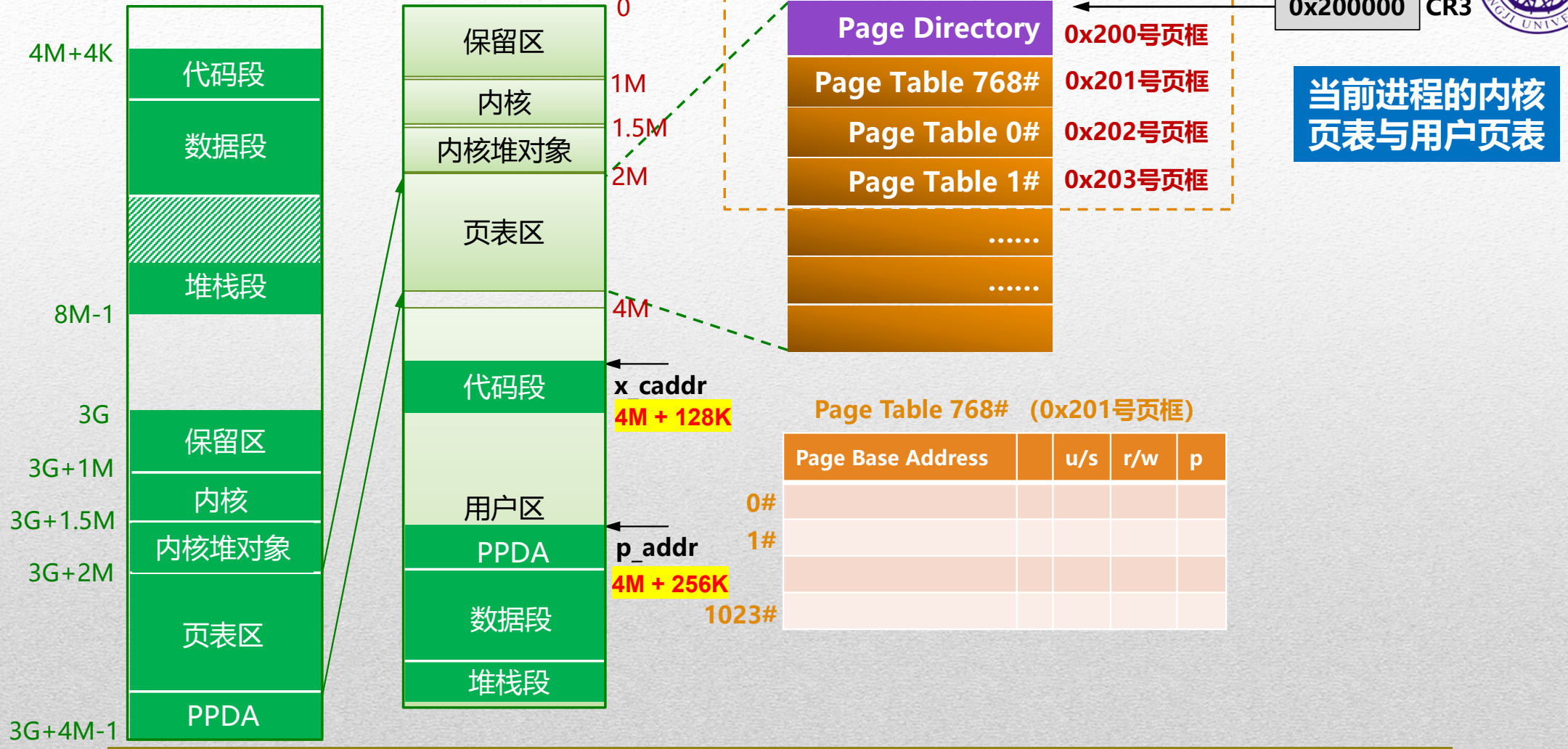


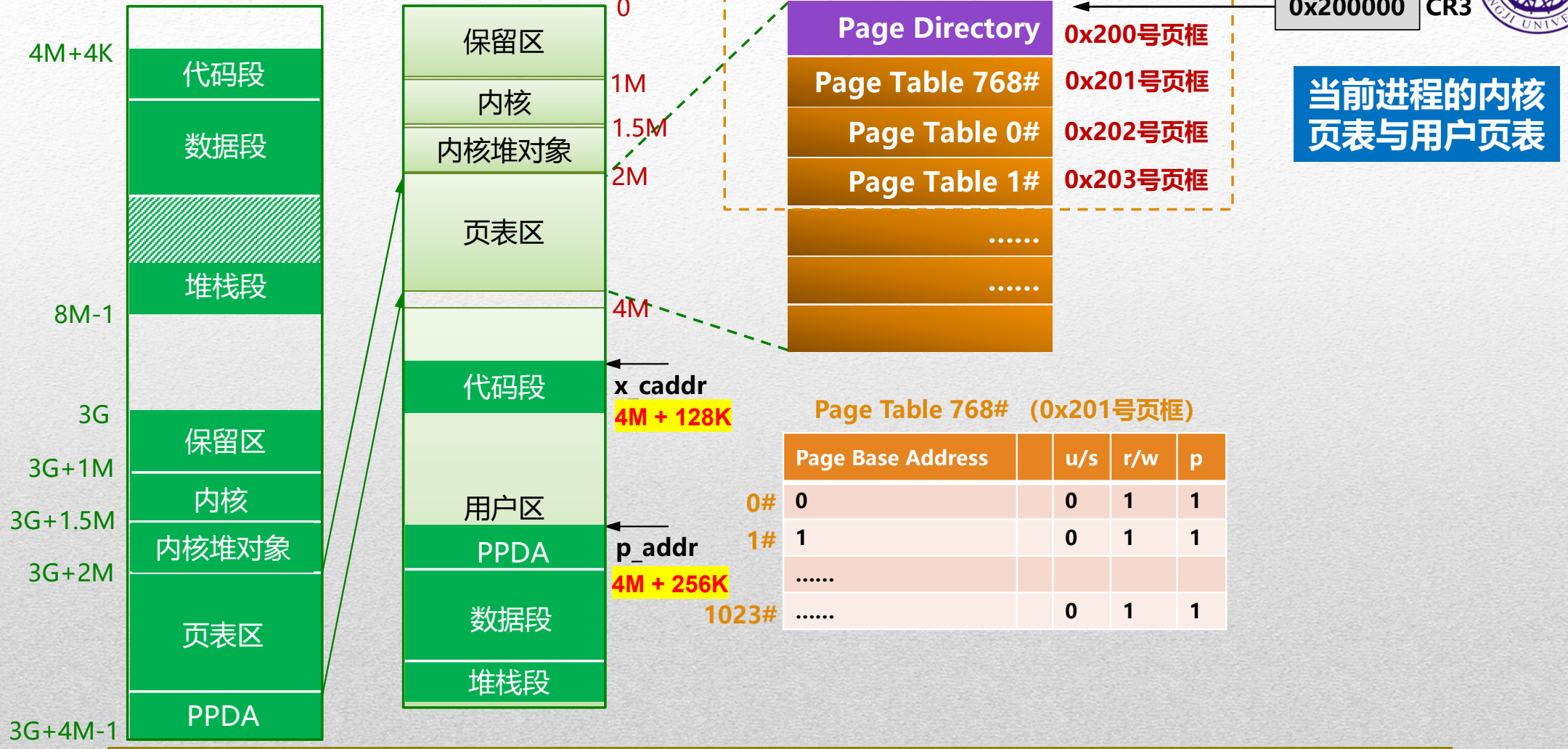
当前进程的内核页表与用户页表

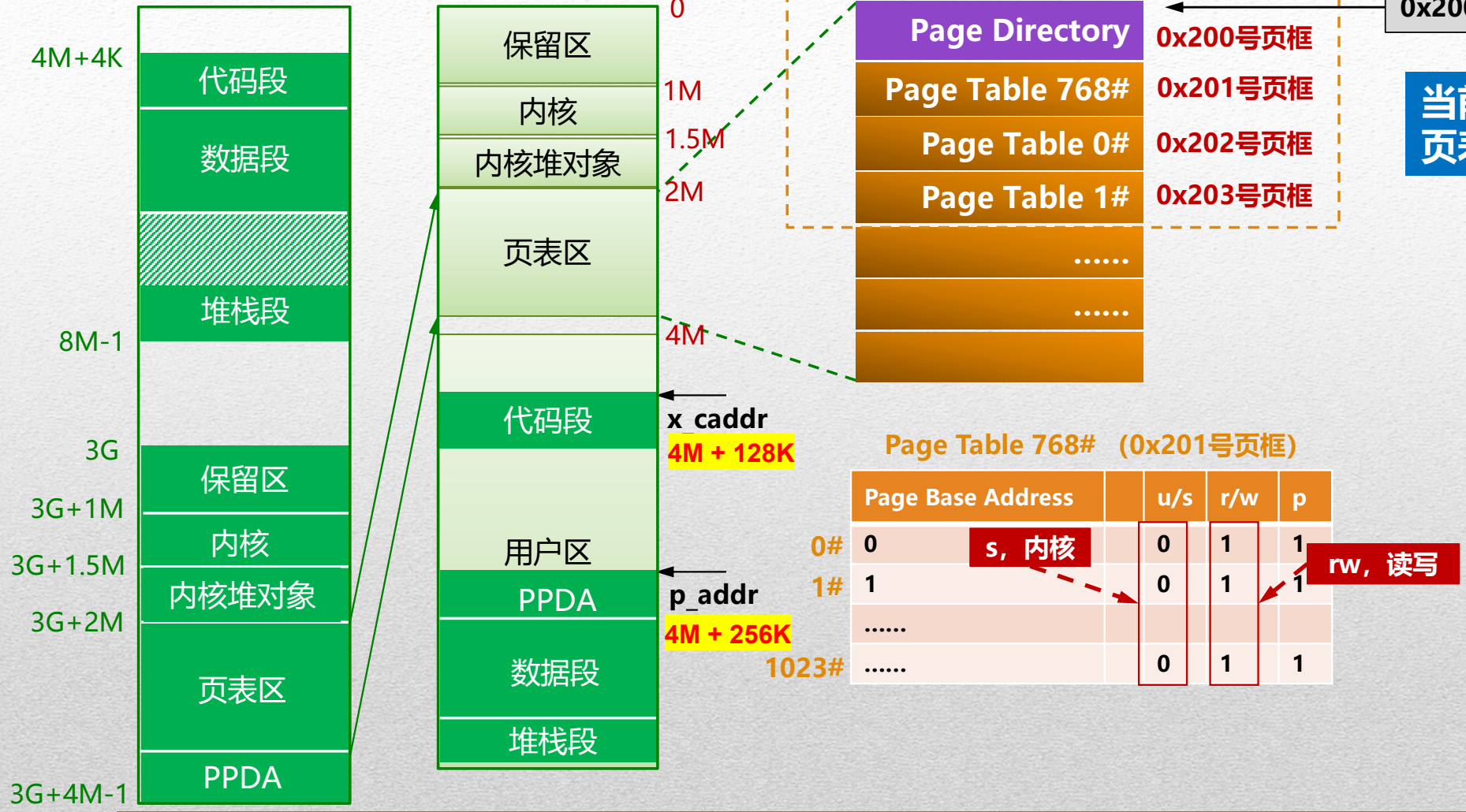
Page Directory（0x200号页框）

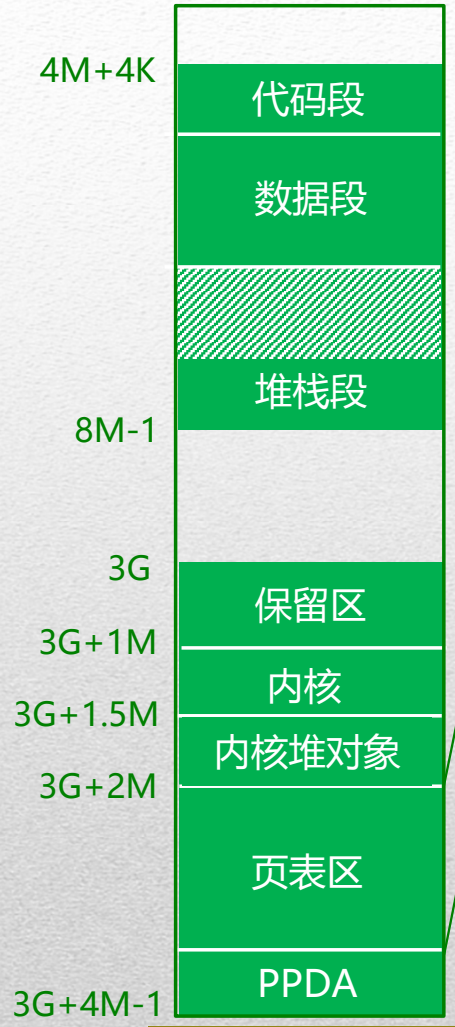
	Page Base Address		u/s	r/w	p
0#	0x202		0	1	1
1#	0x203		0	1	1
	.....				
768#	0x201		0	1	1
	.....				


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 Directory
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
.....
.....
0x200000
CR3
当前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
.....


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
4M + 128K
用户区
PPDA
p_addr
4M + 256K
数据段
堆栈段
0x200000
CR3
Page Directory
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
.....
.....
当前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#


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
保留区
0
内核
1M
内核堆对象
1.5M
2M
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 Directory
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
.....
.....
0x200000
CR3
当前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


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
4M + 128K
用户区
PPDA
p_addr
4M + 256K
数据段
堆栈段
Page Directory
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
.....
.....
0x200000
CR3
当前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
s，内核
0
1
1
1#
1
0
1
1
.....
.....
1023#
.....
0
1
1
rw，读写



x_caddr
4M + 128K

p_addr
4M + 256K

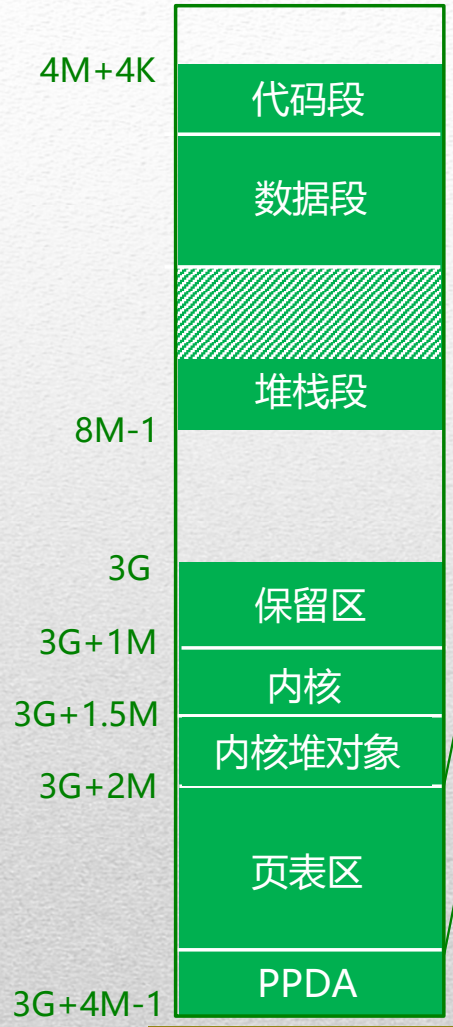
0x200000 CR3

当前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



0
1M
1.5M
2M
4M

x_caddr
4M + 128K

p_addr
4M + 256K

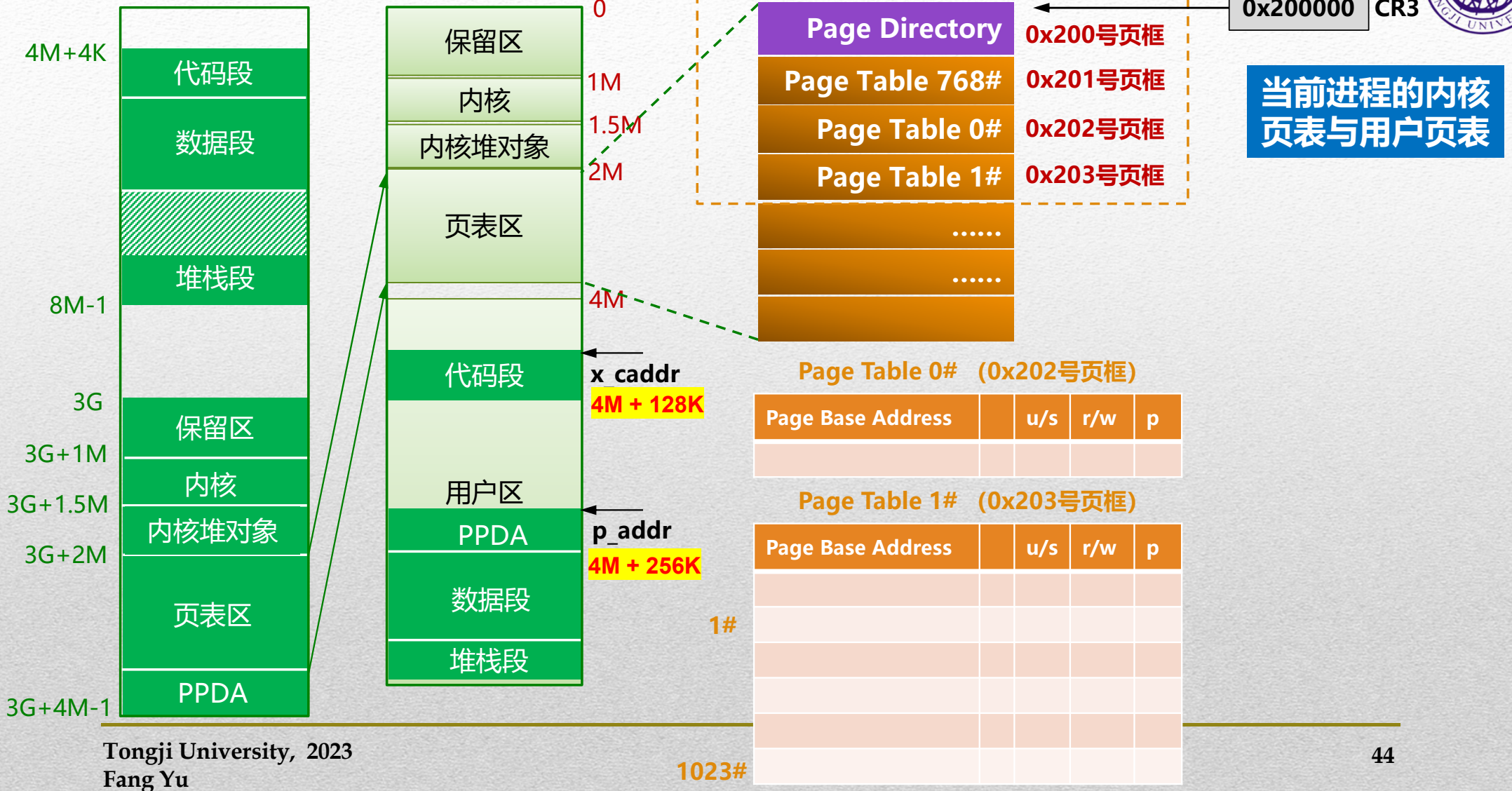
0x200000 CR3

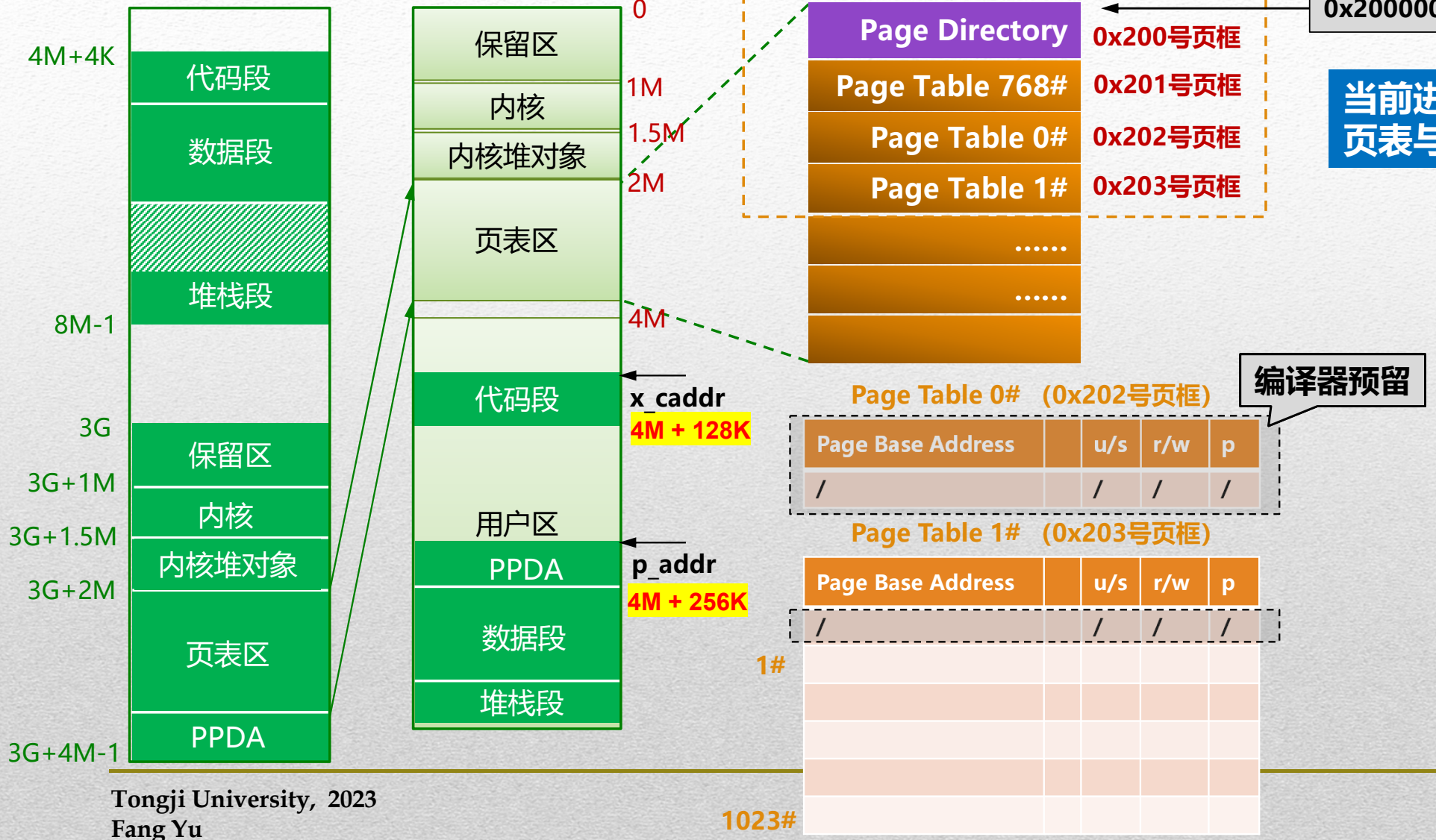
当前进程的内核页表与用户页表

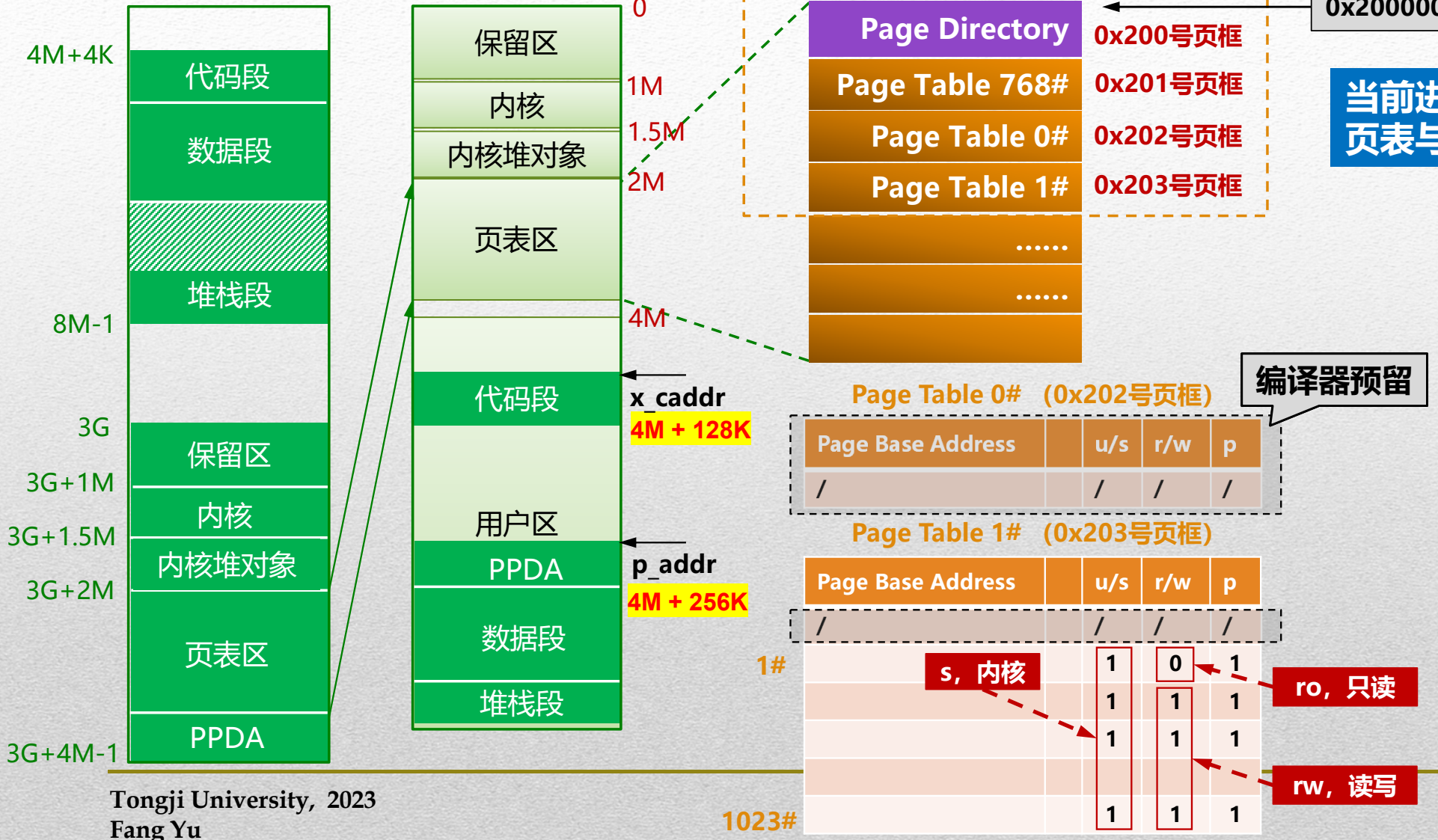
对所有进程都是一样的
(*进程切换调度时不需修改*)

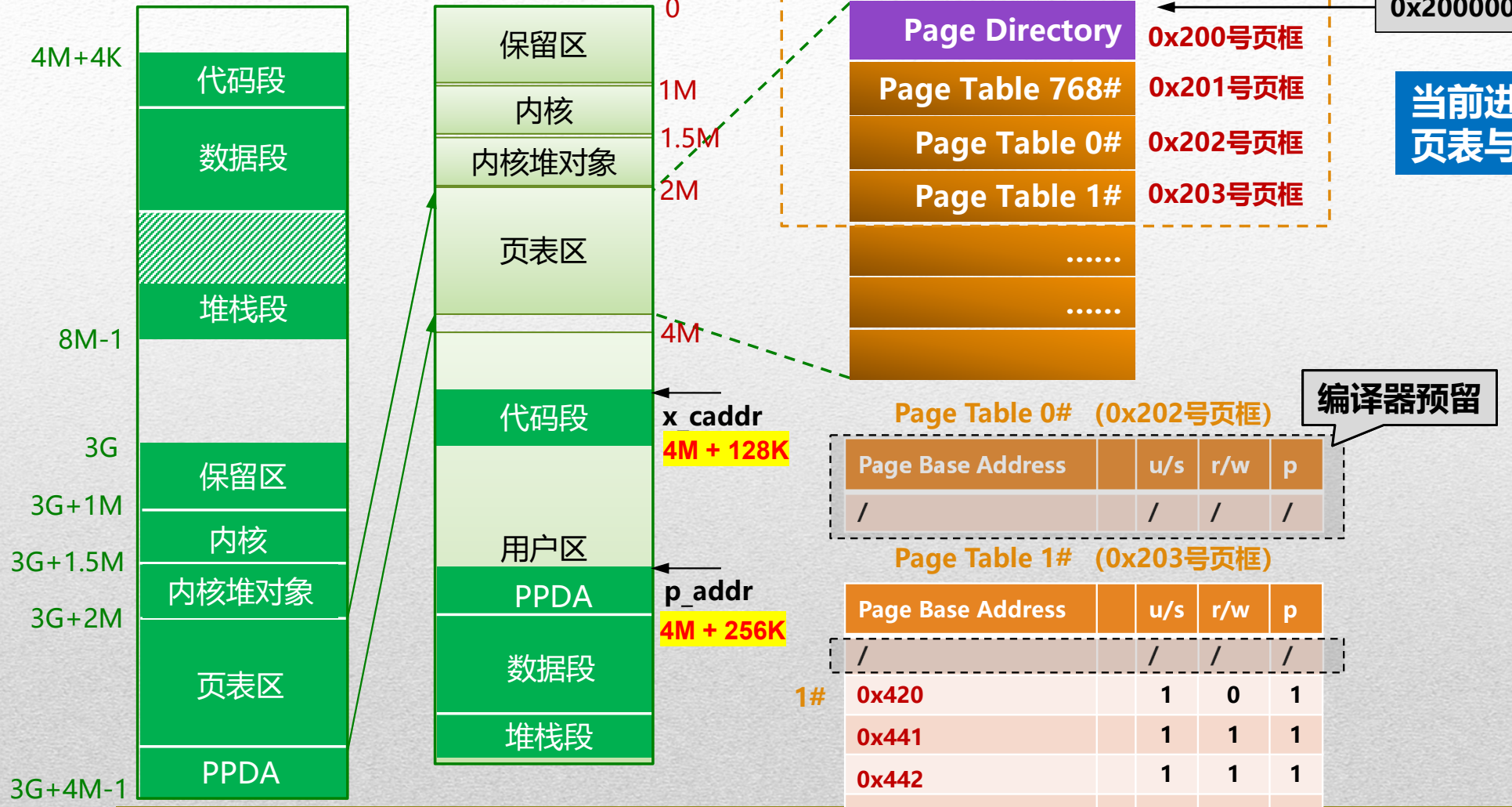
Page Table 768# (0x201号页框)

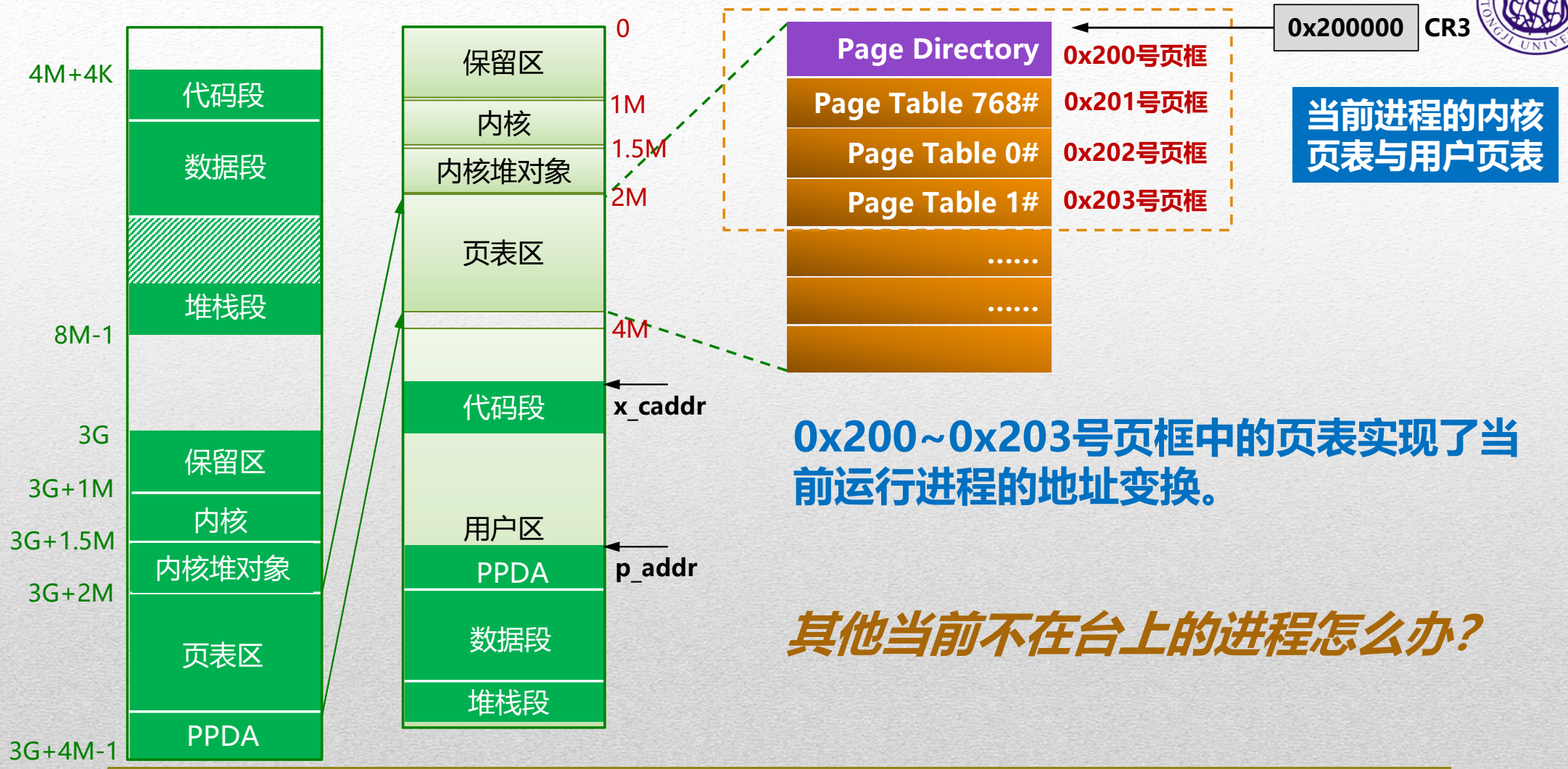
	Page Base Address		u/s	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	.....				
1023#	0x440		0	1	1


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
3G+4M-1
PPDA
0
保留区
1M
内核
1.5M
内核堆对象
2M
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 Directory
0x200000
CR3
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
.....
.....
当前进程的内核页表与用户页表
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#


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
3G+4M-1
PPDA
0
保留区
1M
内核
1.5M
内核堆对象
2M
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
0x200000
CR3
Page Directory
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
.....
.....
当前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
页表区
3G+4M-1
PPDA
0
保留区
1M
内核
1.5M
内核堆对象
2M
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 Directory
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
.....
.....
0x200000
CR3
当前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
ro，只读
1
1
1
1
1
1
rw，读写
1023#
1
1
1


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
3G+4M-1
PPDA
保留区
内核
内核堆对象
页表区
代码段
用户区
PPDA
数据段
堆栈段
0
1M
1.5M
2M
4M
x_caddr
4M + 128K
p_addr
4M + 256K
Page Directory
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
0x200000
CR3
当前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



当前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。

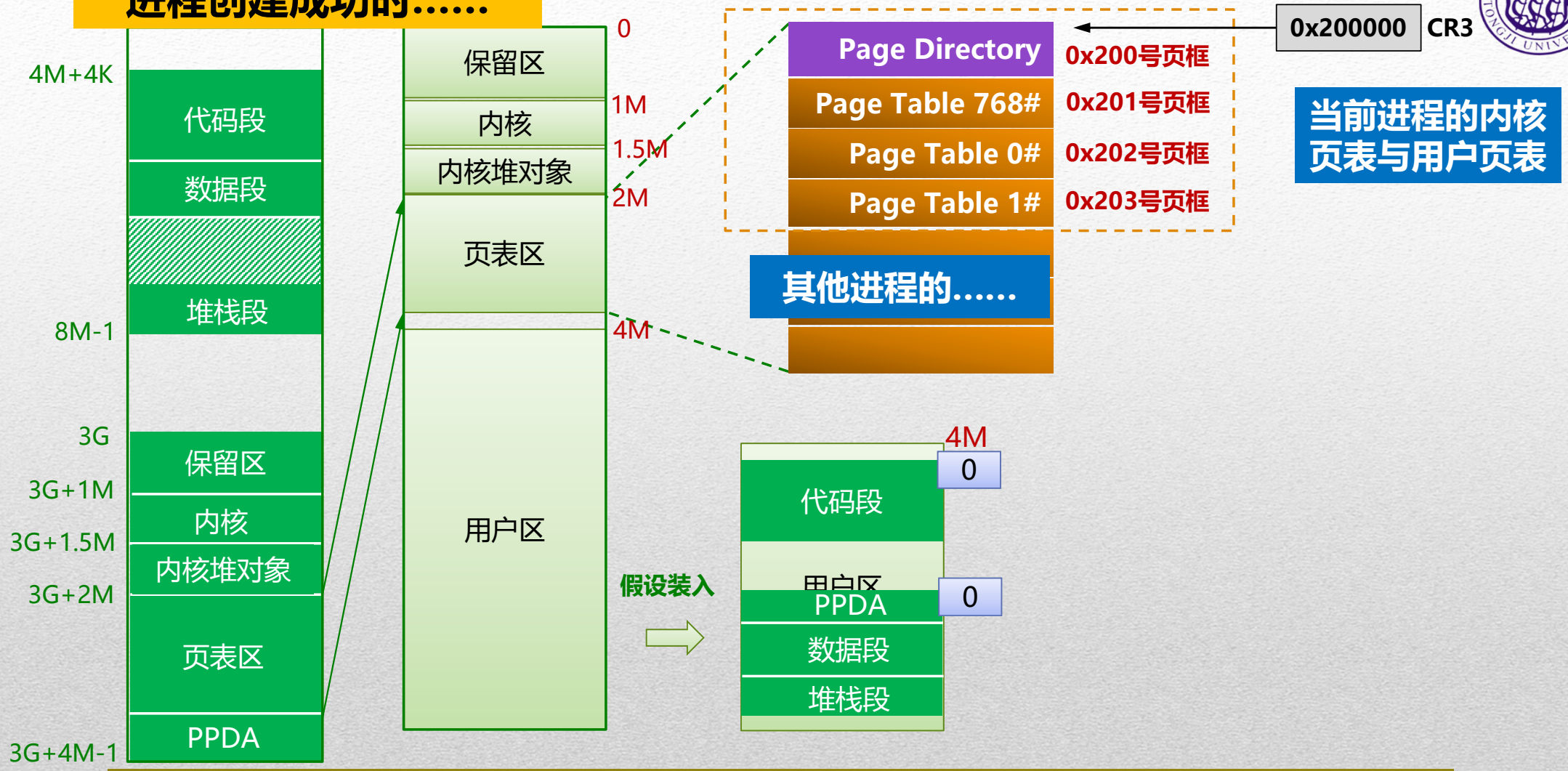
其他当前不在台上的进程怎么办?

当前进程的内核页表与用户页表

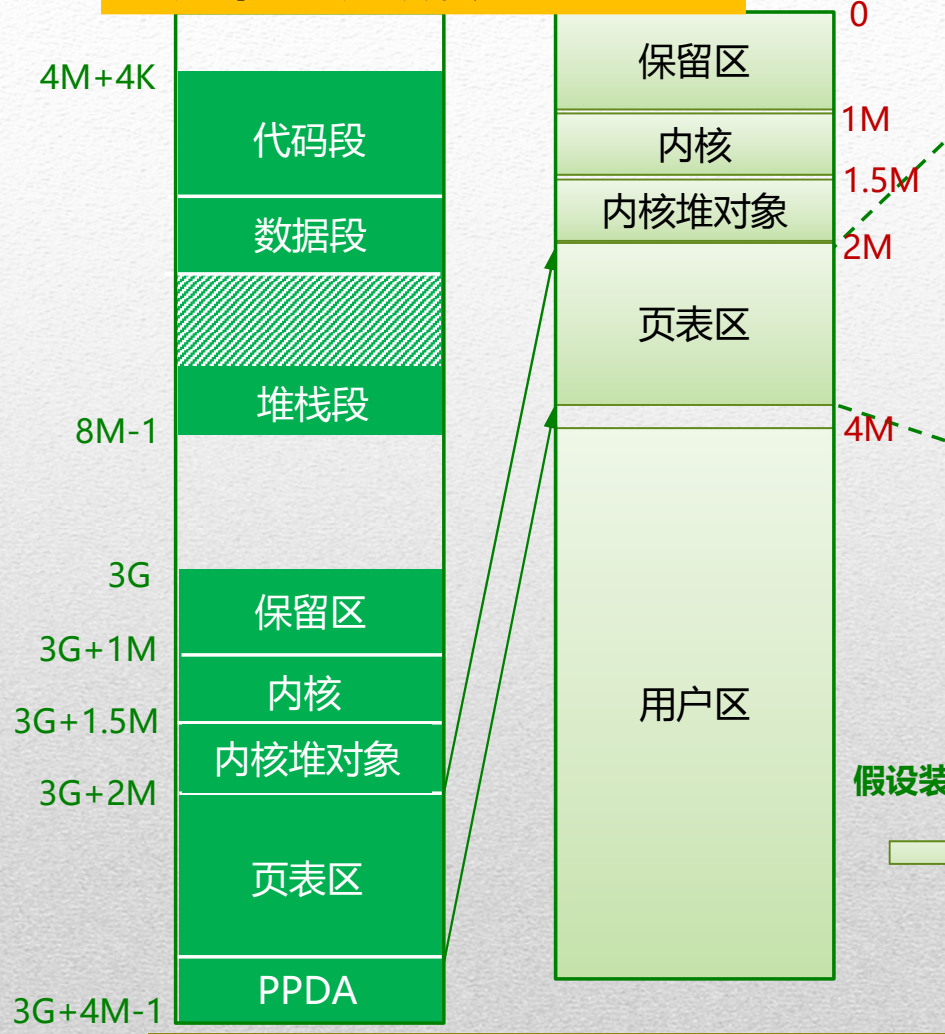
0x200~0x203号页框中的页表实现了当前运行进程的地址变换。

其他当前不在台上的进程怎么办?

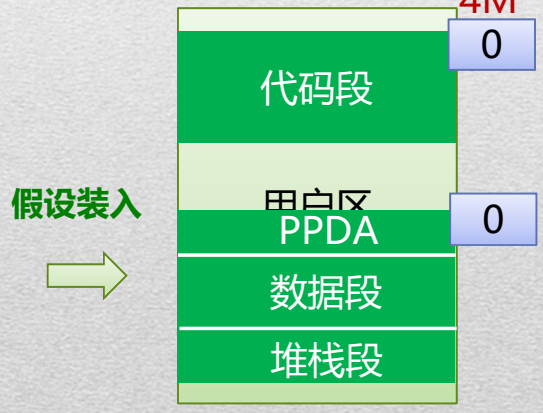
进程创建成功时……


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
用户区
4M
Page Directory
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
其他进程的……
0x200000
CR3
当前进程的内核
页表与用户页表
假设装入
4M
0
代码段
用户区
PPDA
0
数据段
堆栈段

进程创建成功时……



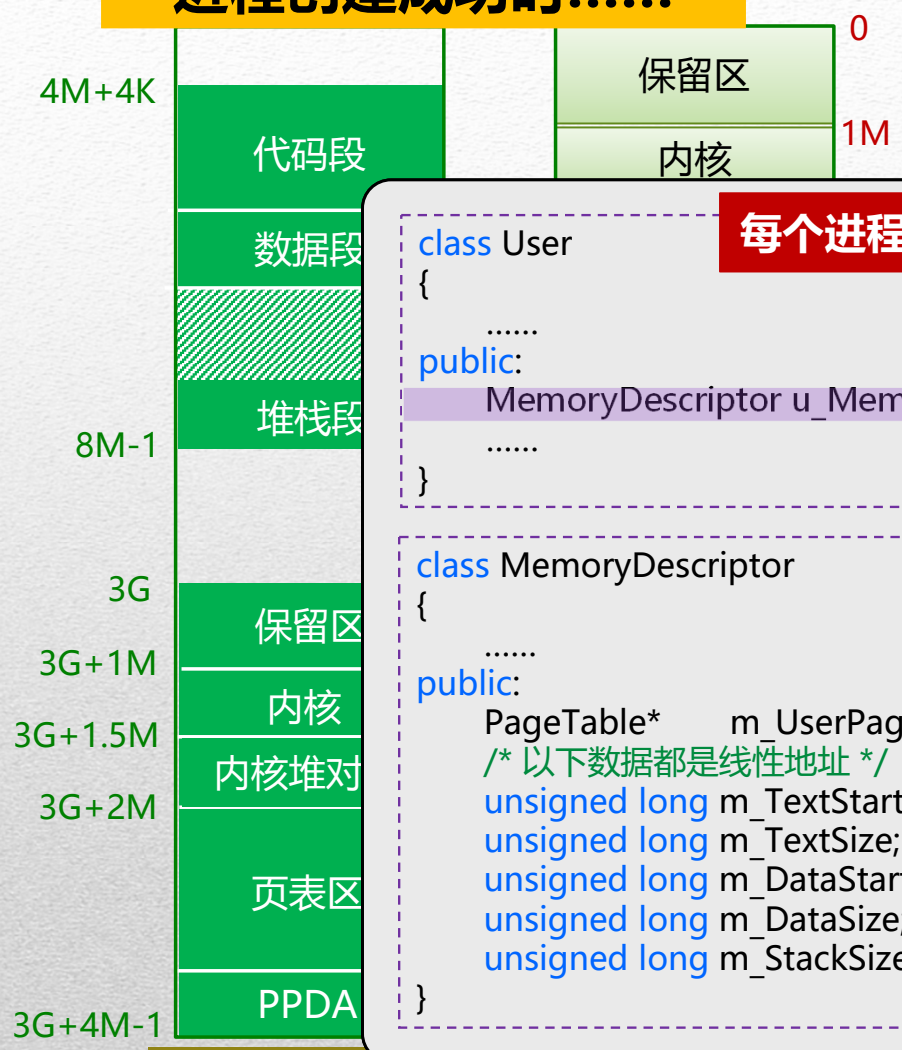
当前进程的内核页表与用户页表



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

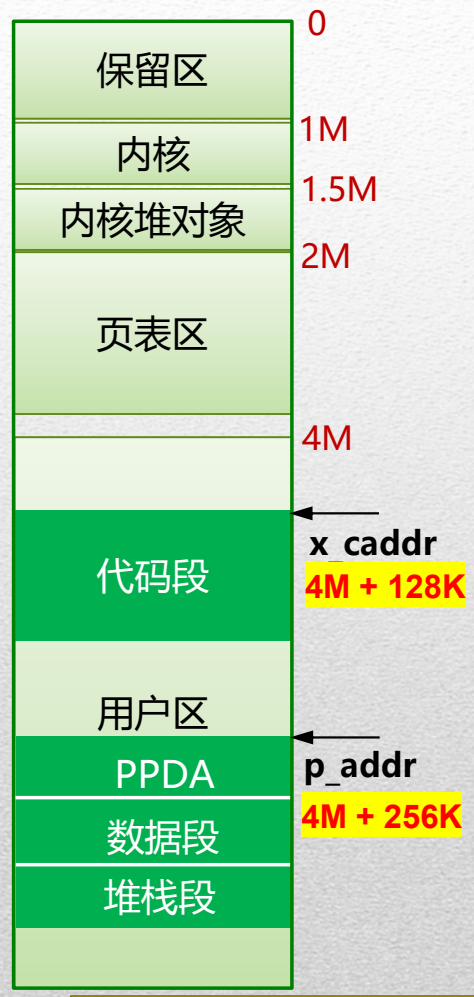
```
class User
{
        ......
public:
        MemoryDescriptor u_MemoryDescriptor;
        ......
}

class MemoryDescriptor
{
        ......
public:
        PageTable*      m_UserPageTableArray;
        /* 以下数据都是线性地址 */
        unsigned long m_TextStartAddress;
        unsigned long m_TextSize;
        unsigned long m_DataStartAddress;
        unsigned long m_DataSize;
        unsigned long m_StackSize;
}
```

当前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

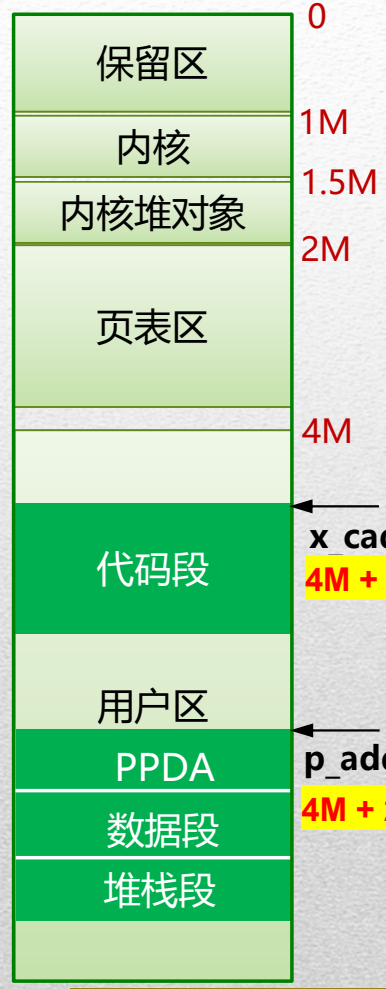
进程被调度上台时……



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

进程被调度上台时……



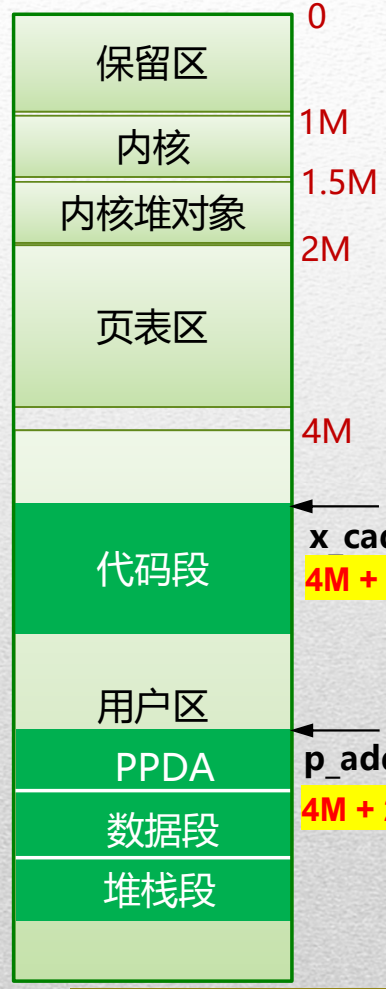
Page Directory（0x200号页框）

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

	Page 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

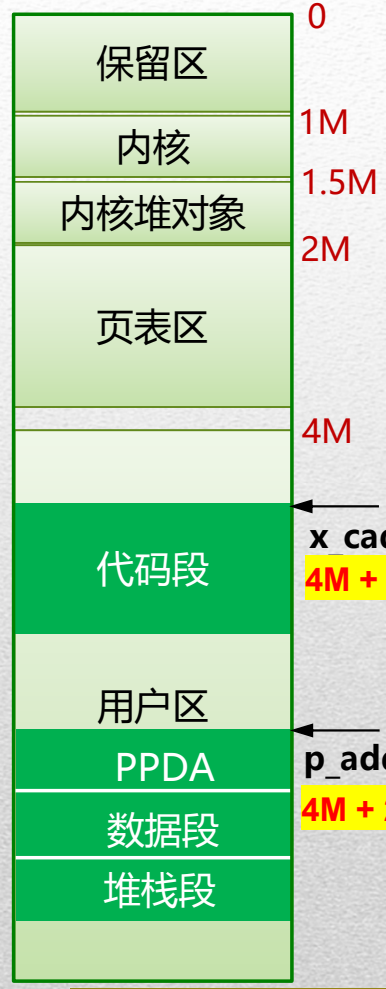
Page Table 0#（0x202号页框）

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……



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#	②. if r/w==0, x_caddr>>12，加上PBA全0				
2047#	2		1	1	1

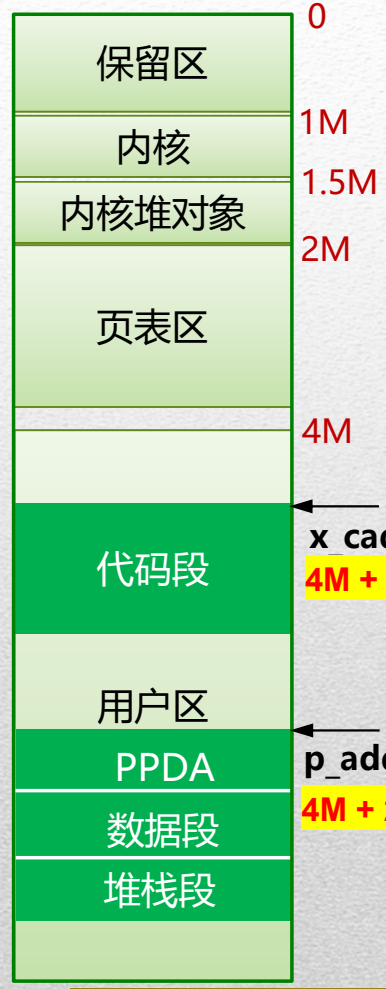
Page Table 0#（0x202号页框）

	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
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
	……				

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 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）

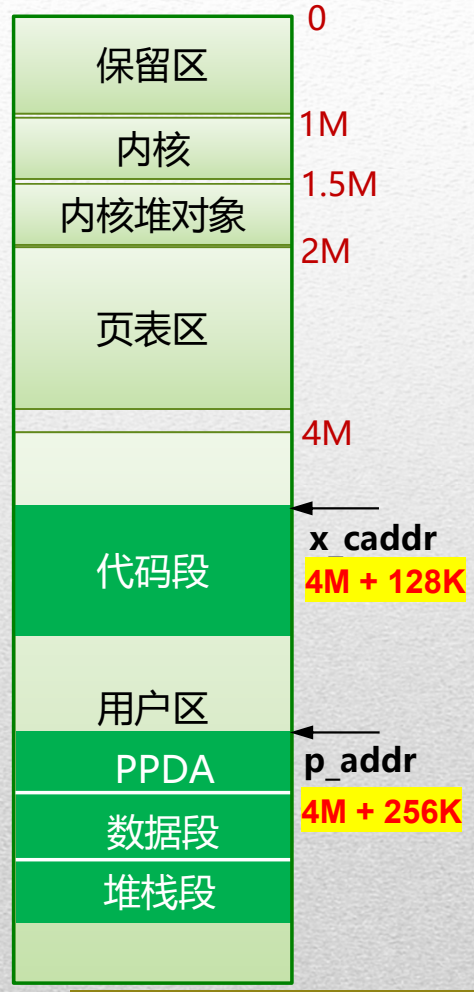
Page Base Address		u/s	r/w	p

Page Table 1#（0x203号页框）

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

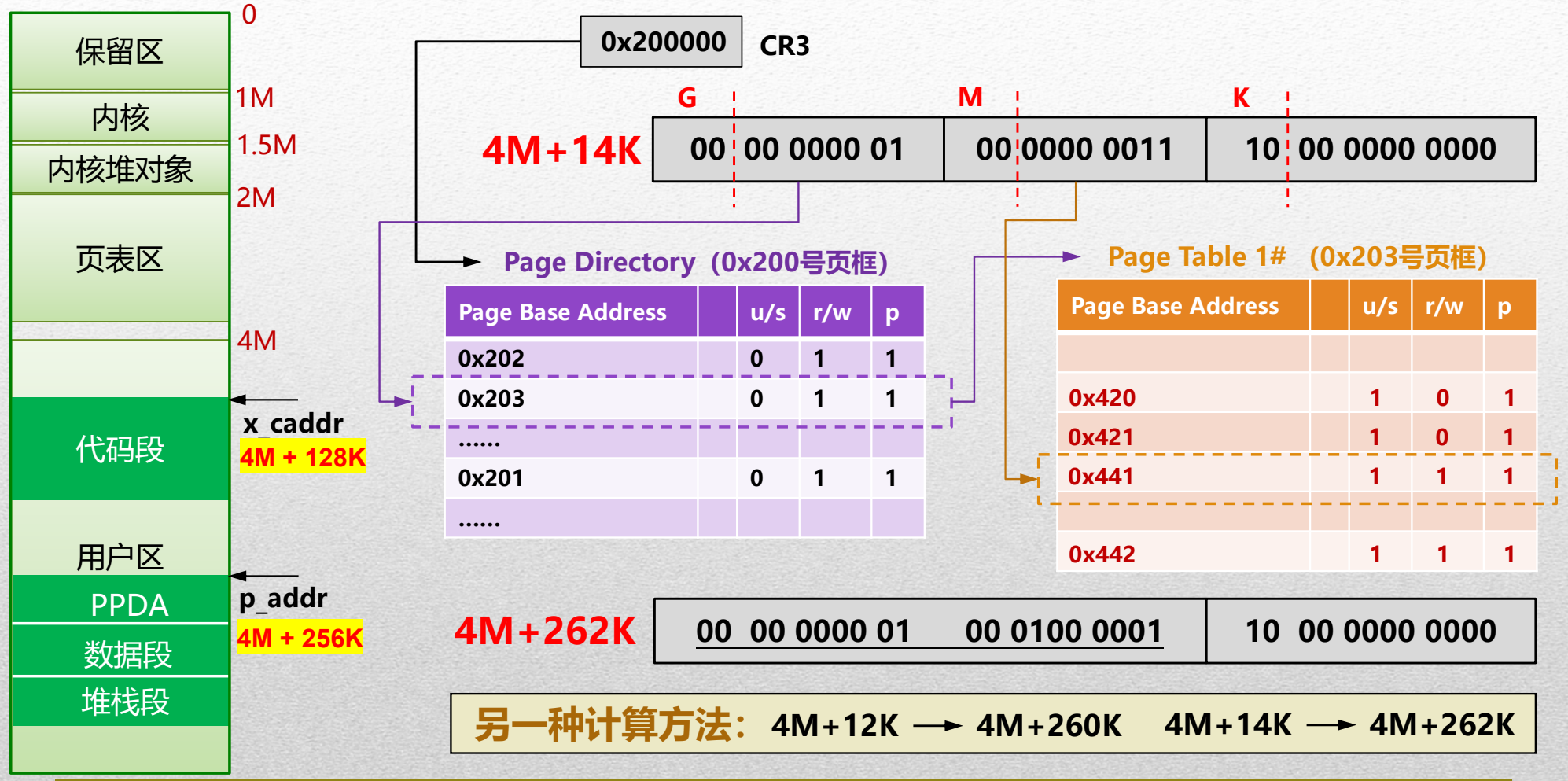
Page Base Address		u/s	r/w	p
0x420		1	0	1
0x421		1	0	1
0x441		1	1	1
0x442		1	1	1

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]



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 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 V6++中对存储空间的管理

内核对象区、页表区、用户区均采用可变分区方式进行分配，分别由三张空闲分区表管理：

```
public:MapNode map[512];  最多包含512个表项的空闲分区表

struct MapNode
{
    unsigned long m_Size;
    unsigned long m_AddressIdx;
};
```

其中每一个表项包含空闲分区的大小和分区的起始地址

内核堆对象区，其map初始化为：

`map[0]. m_AddressIdx = 1.5M, map[0]. m_Size = 0.5M`。

负责管理页表区，其map初始化为：

`map[0]. m_AddressIdx = 2M+16K`，即：从204号页框开始。

`map[0]. m_Size = 2M-16K`，即：剩余的页表区。

用户区，其map初始化为：

`map[0]. m_AddressIdx = 4M`，大小至整个物理内存。

大小	起始地址
0.5M	1.5M

大小	起始地址
2M-16K	2M+16K

大小	起始地址
至整个物理内存	4M

UNIX V6++中对存储空间的管理

KernelAllocator，KernelPageManager，UserPageManager三个类中各有一对空闲区分配和回收的函数

回收一个大小为size，起始地址为addIdx的分区到map中

按首次适应算法在map中分配一个大小为size的分区

```
public:

unsigned long Alloc(MapNode map[], unsigned long size);
unsigned long Free(MapNode map[], unsigned long size,
                                  unsigned long addrIdx);
```

分配时

KernelAllocator:
size = 实际需求

按首次适应算法分配

KernelPageManager:
size = 两个页框（8K）

UserPageManager:
size = 向上取整（实际需求/4k）× 4k
即：满足需求的4K整数倍。

回收时

KernelAllocator:
size = 实际占用

考虑四种回收分区的位置情况

KernelPageManager:
size = 两个页框（8K）

UserPageManager:
size = 实际占用

进程创建时：

在页表区中申请**两个空闲页框**，建相对虚实地址映射表

1

KernelPageManager::Alloc(map[], 8K);

和父进程共享已经在内存的代码段：

实际需求 = 可交换部分

代码段与可交换部分不连续

按进程**实际需求**在用户区中申请内存

2

UserPageManager::Alloc(map[], 向上取整（实际需求/4k）× 4k);

进程图象交换时：

最后一个使用该代码段的进程：
一起释放代码段
否：不释放

换出进程释放曾经占用的内存

1

UserPageManager::Free(map[], p_size, p_addr);
UserPageManager::Free(map[], x_size, x_caddr);

第一个使用该代码段的进程：

实际需求 = 代码段 + 可交换部分

代码段与可交换部分连续

共享一个已经在内存的代码段：

实际需求 = 可交换部分

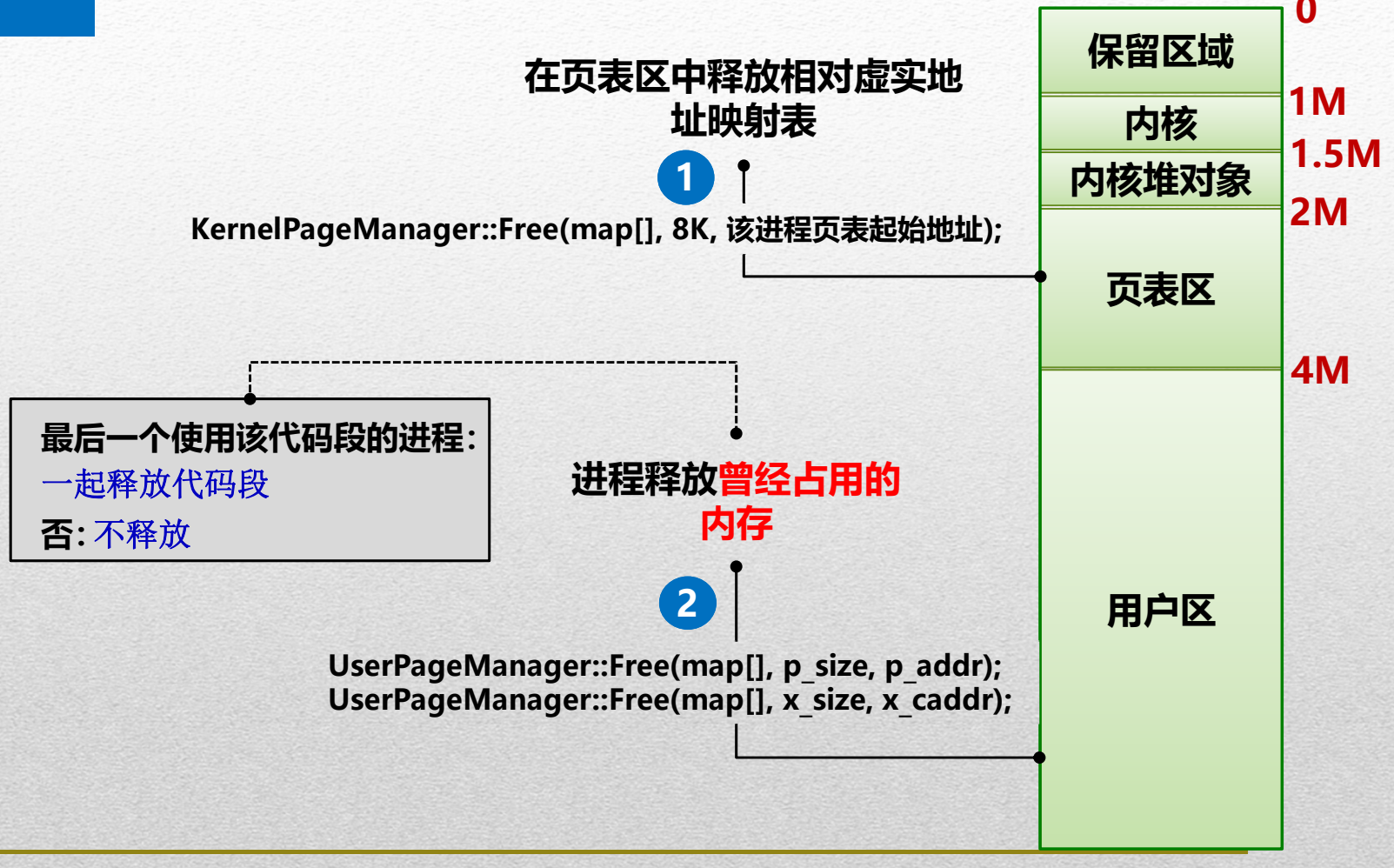
代码段与可交换部分不连续

为换入进程实际需求在用户区中申请内存

2

UserPageManager::Alloc(map[], 向上取整（实际需求/4k） × 4k);

进程终止时：


在页表区中释放相对虚实地址映射表
1
KernelPageManager::Free(map[], 8K, 该进程页表起始地址);
最后一个使用该代码段的进程：
一起释放代码段
否：不释放
进程释放曾经占用的内存
2
UserPageManager::Free(map[], p_size, p_addr);
UserPageManager::Free(map[], x_size, x_caddr);
保留区域
内核
内核堆对象
页表区
用户区
0
1M
1.5M
2M
4M

Process类

	名称	类型	含义
进程标识	p_uid	short	用户ID
	p_pid	int	进程标识数，进程编号
	p_ppid	int	父进程标识数
进程图象在内存中的位置信息	p_addr	unsigned long	ppda区在物理内存中的起始地址
	p_size	unsigned int	进程图象（除代码段以外部分）的长度，以字节单位
	p_textp	Text *	指向该进程所运行的代码段的描述符
进程调度相关信息	p_stat	ProcessState	进程当前的调度状态
	p_flag	int	进程标志位，可以将多个状态组合
	p_pri	int	进程优先数
	p_cpu	int	cpu值，用于计算p_pri
	p_nice	int	进程优先数微调参数
	p_time	int	进程在盘交换区上（或内存内）的驻留时间
	p_wchan	unsigned long	进程睡眠原因
信号与控制台终端	p_sig	int	进程信号
	p_ttyp	TTy*	进程tty结构地址

User类

	名称	类型	含义
进程的用户标识	u_uid	short	有效用户ID
	u_gid	short	有效组ID
	u_ruid	short	真实用户ID
	u_rgid	short	真实组ID
进程的时间相关	u_utime	int	进程用户态时间
	u_stime	int	进程核心态时间
	u_cutime	int	子进程用户态时间总和
	u_cstime	int	子进程核心态时间总和
现场保护相关	u_rsav[2]	unsigned long	用于保存esp与ebp指针
	u_ssav[2]	unsigned long	用于对esp和ebp指针的二次保护
内存管理相关	*u_procp	Process	指向该u结构对应的Process结构
	u_MemoryDescriptor	MemoryDescriptor	封装了进程的图象在内存中的位置、大小等信息
系统调用相关	EAX = 0	static const int	访问现场保护区中EAX寄存器的偏移量
	*u_ar0	unsigned int	指向核心栈现场保护区EAX寄存器存放的栈单元
	u_arg[5];	int	存放当前系统调用参数
	*u_dirp	char	系统调用参数（一般用于Pathname）的指针

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++对存储空间的管理

E05：存储管理（UNIX V6++进程图象）