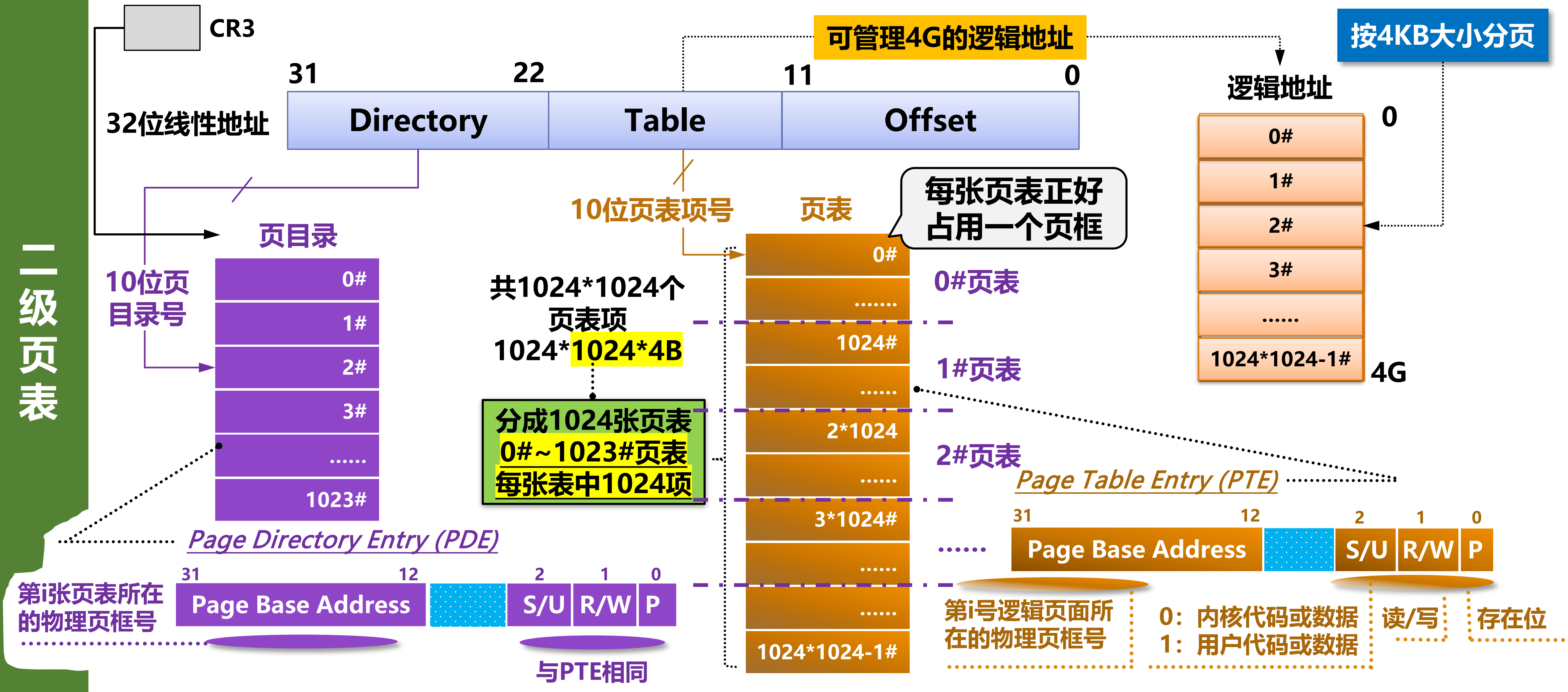
第三章

存储管理

i386线性地址空间



二级页表
CR3
可管理4G的逻辑地址
按4KB大小分页
31
22
11
0
32位线性地址
Directory
Table
Offset
逻辑地址
0
0#
1#
2#
3#
......
1024*1024-1#
4G
10位页表项号
页表
每张页表正好占用一个页框
页目录
10位页目录号
0#
1#
2#
3#
......
1023#
共1024*1024个页表项
1024*1024*4B
分成1024张页表
0#~1023#页表
每张表中1024项
0#
......
1024#
......
2*1024
......
3*1024#
......
......
1024*1024-1#
0#页表
1#页表
2#页表
Page Table Entry (PTE)
31
12
2
1
0
......
Page Base Address
S/U
R/W
P
第i号逻辑页面所在的物理页框号
0：内核代码或数据
1：用户代码或数据
读/写
存在位
Page Directory Entry (PDE)
31
12
2
1
0
第i张页表所在的物理页框号
Page Base Address
S/U
R/W
P
与PTE相同

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

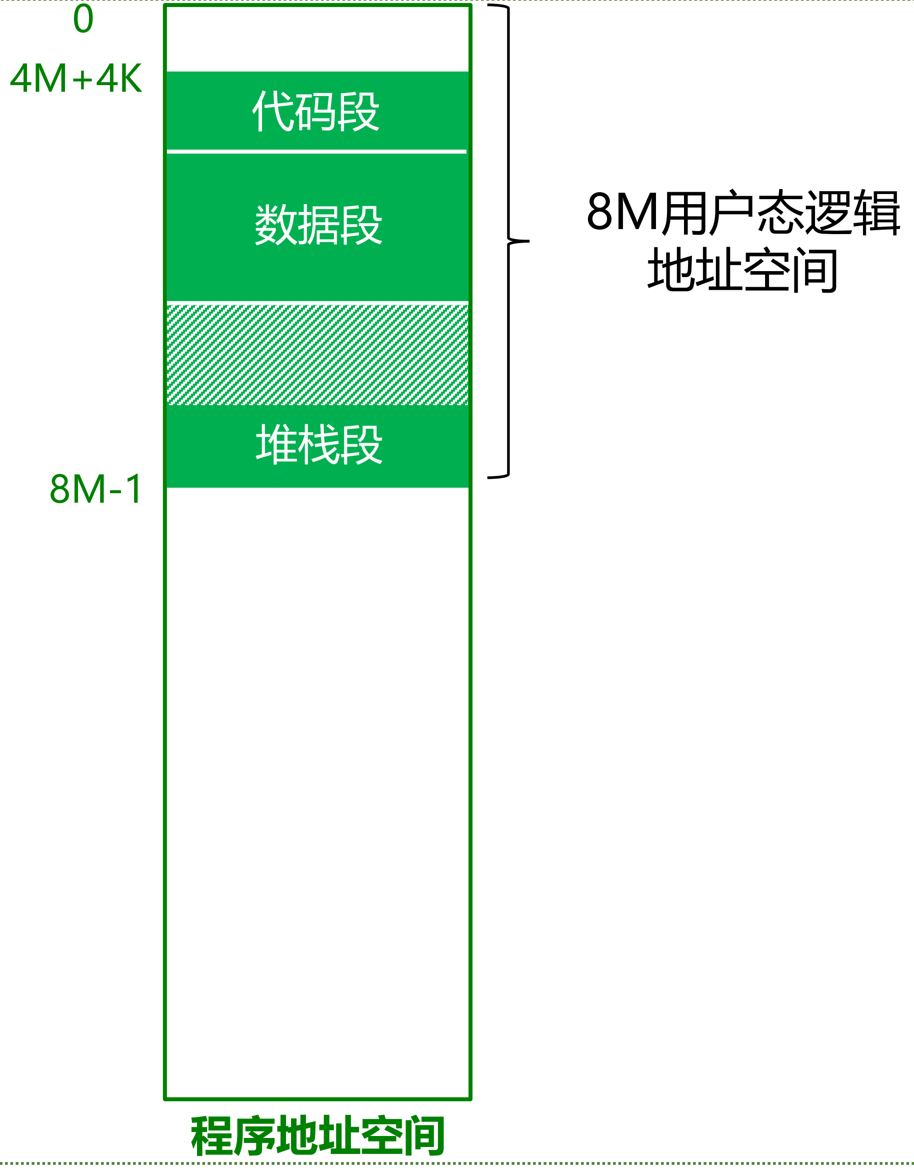
3.4　段式与段页式存储管理**

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

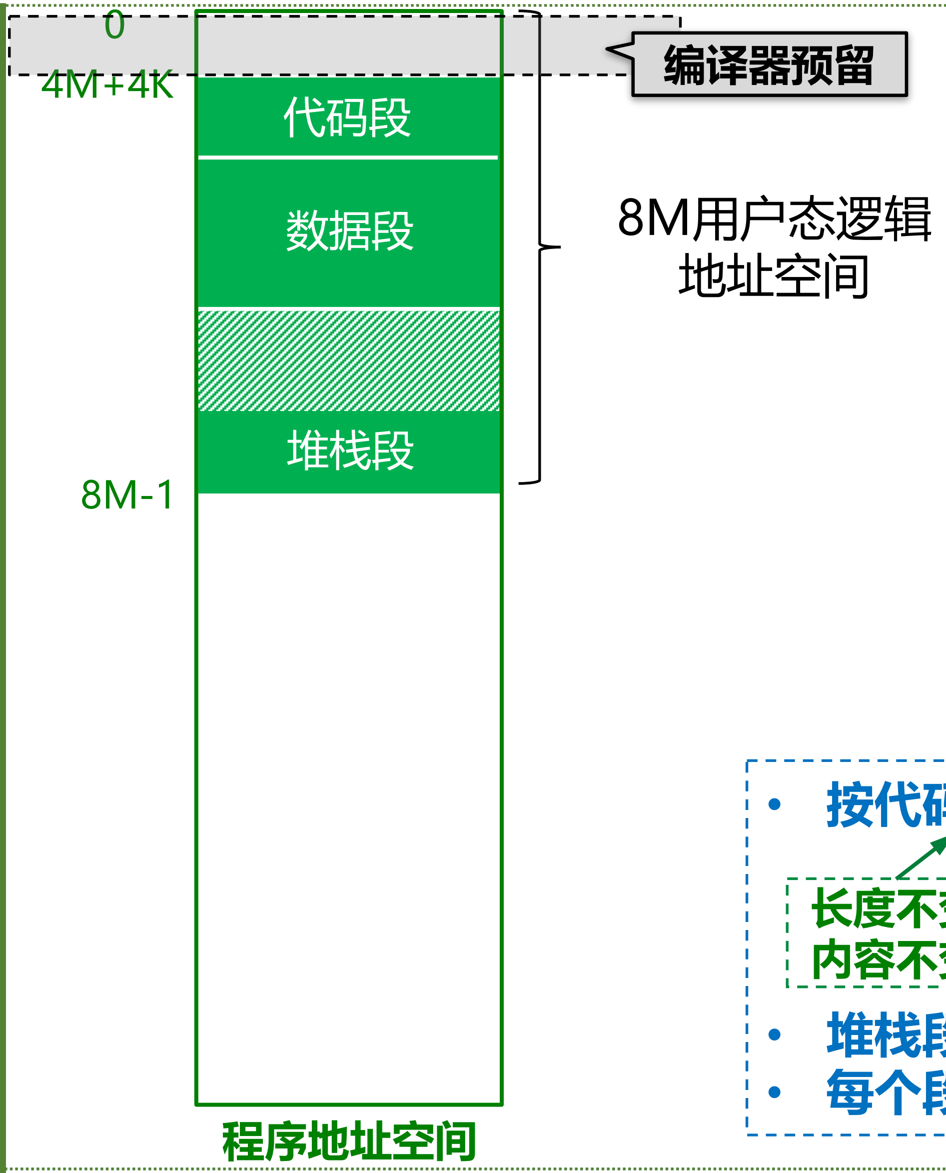
UNIX V6++的地址变换

页表的构成



UNIX V6++的地址变换

页表的构成



- 按代码段、数据段和堆栈段的顺序

长度不变 内容不变 | 长度变 内容变 | 长度变 内容变

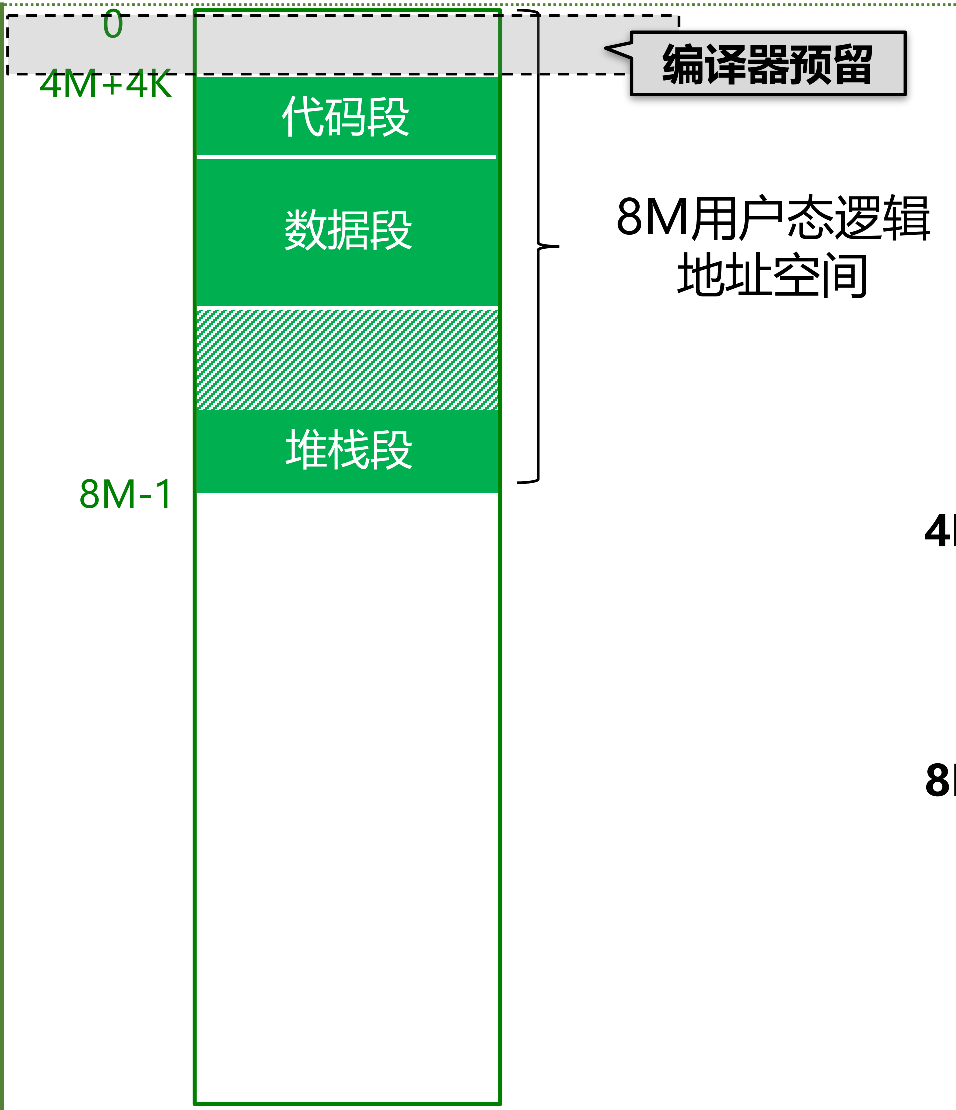
用户程序地址空间

- 堆栈段从高地址向低地址生长
- 每个段落长度都是4K的整数倍

默认条件下，分配一张初始堆栈页面（第2047#逻辑页）。随着程序嵌套调用的层次增加，可动态追加新的页面。

UNIX V6++的地址变换

页表的构成

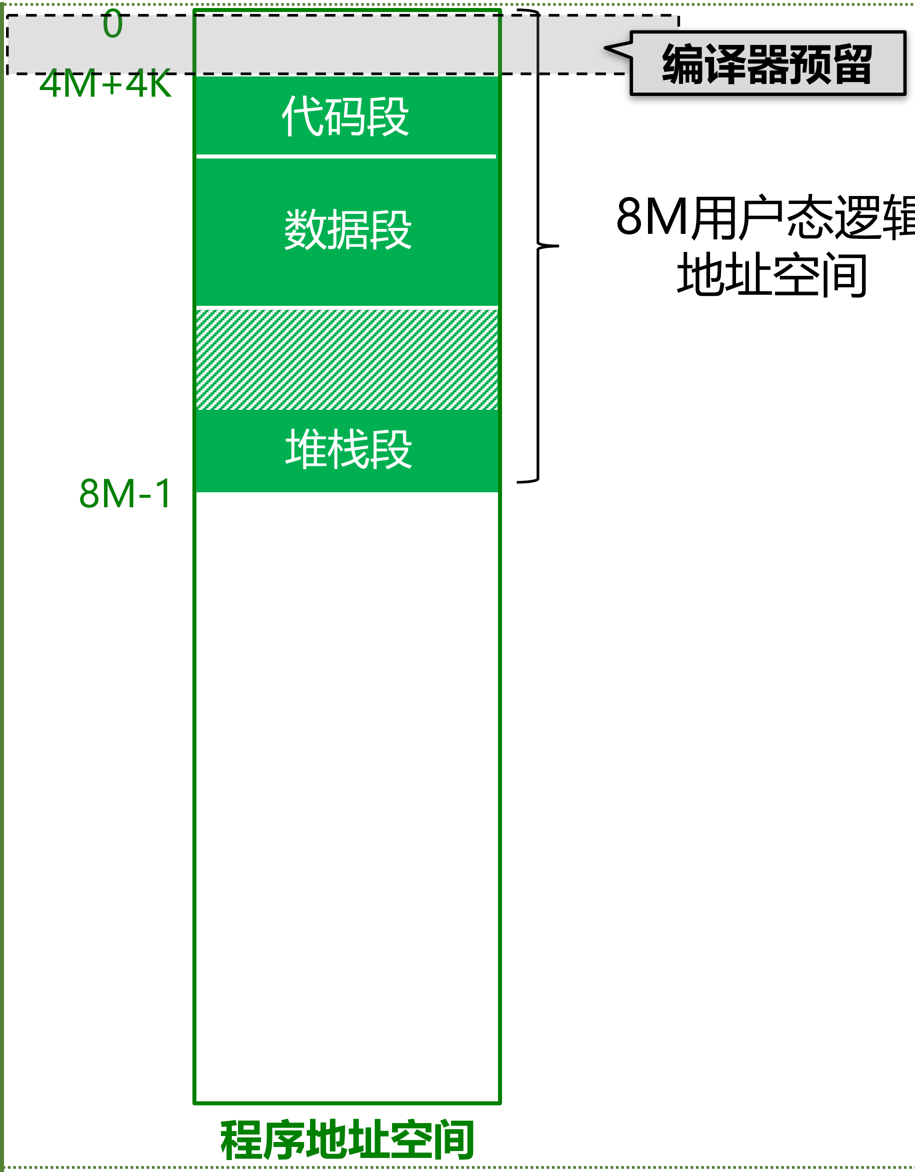


程序地址空间

	31 ~ 22	11 ~ 0	
	Directory	Table	Offset
	G	M	K
0	00 00 0000 00	00 0000 0000	00 00 0000 0000
~			
4M-1	00 00 0000 00	11 1111 1111	11 11 1111 1111
4M	00 00 0000 01	00 0000 0000	00 00 0000 0000
~			
8M-1	00 00 0000 01	11 1111 1111	11 11 1111 1111

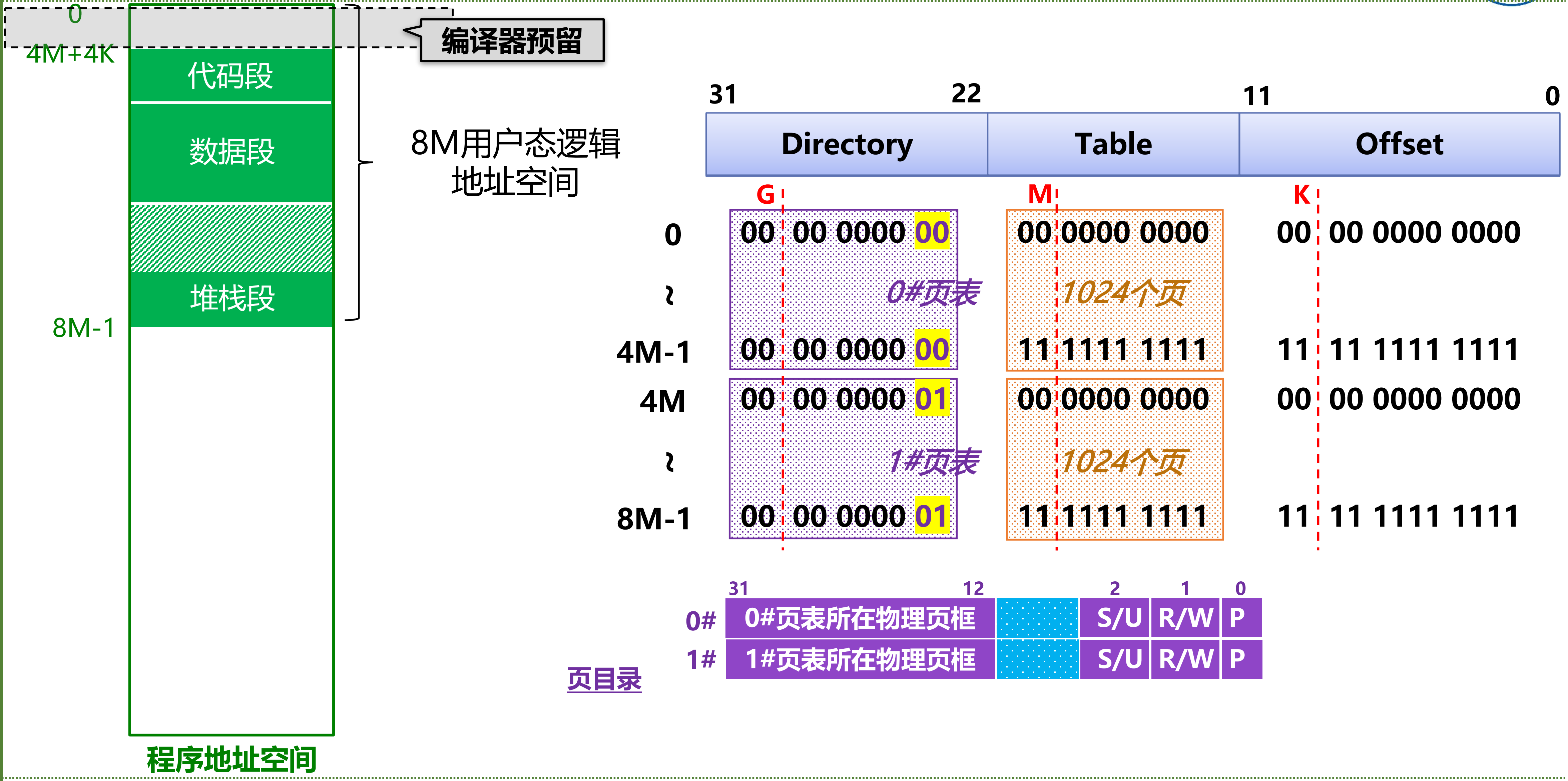
UNIX V6++的地址变换

页表的构成



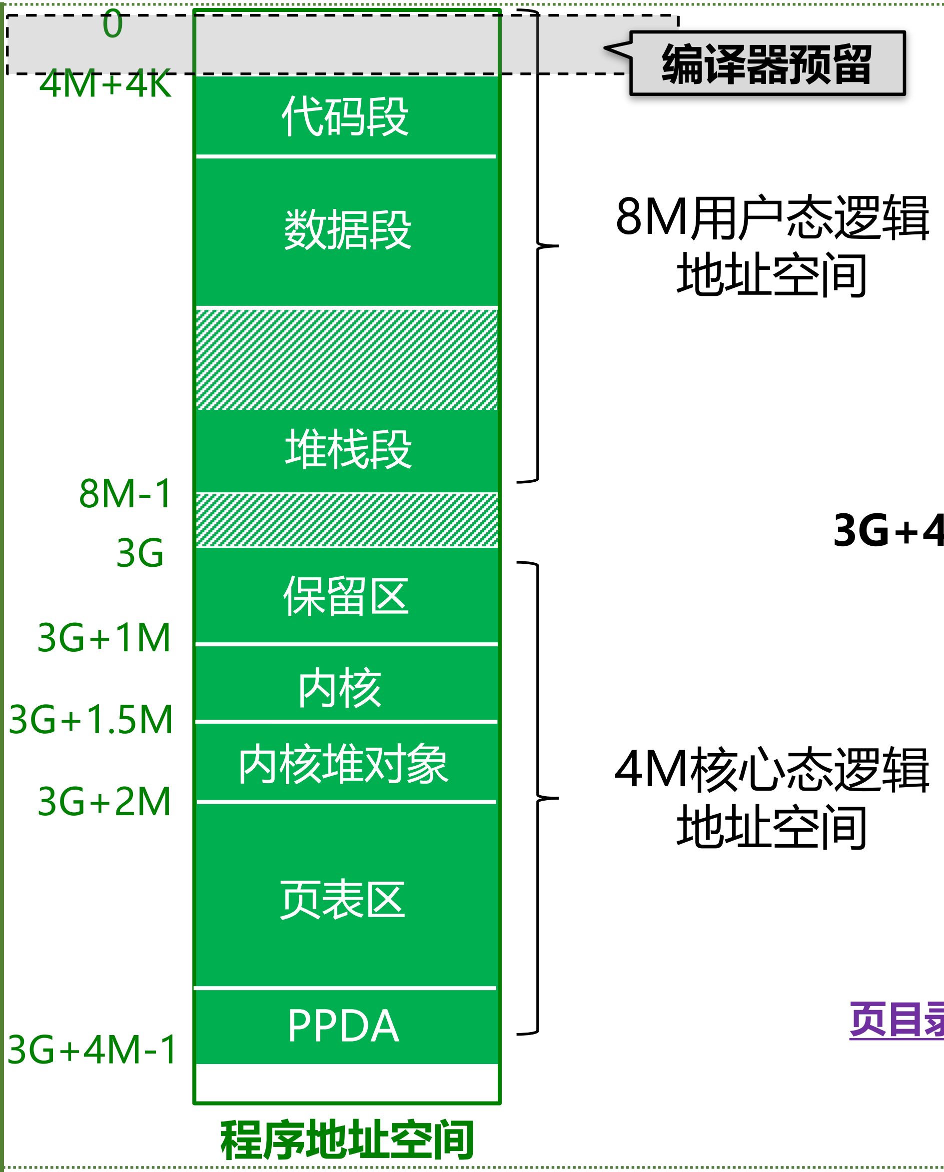
UNIX V6++的地址变换

页表的构成



UNIX V6++的地址变换

页表的构成



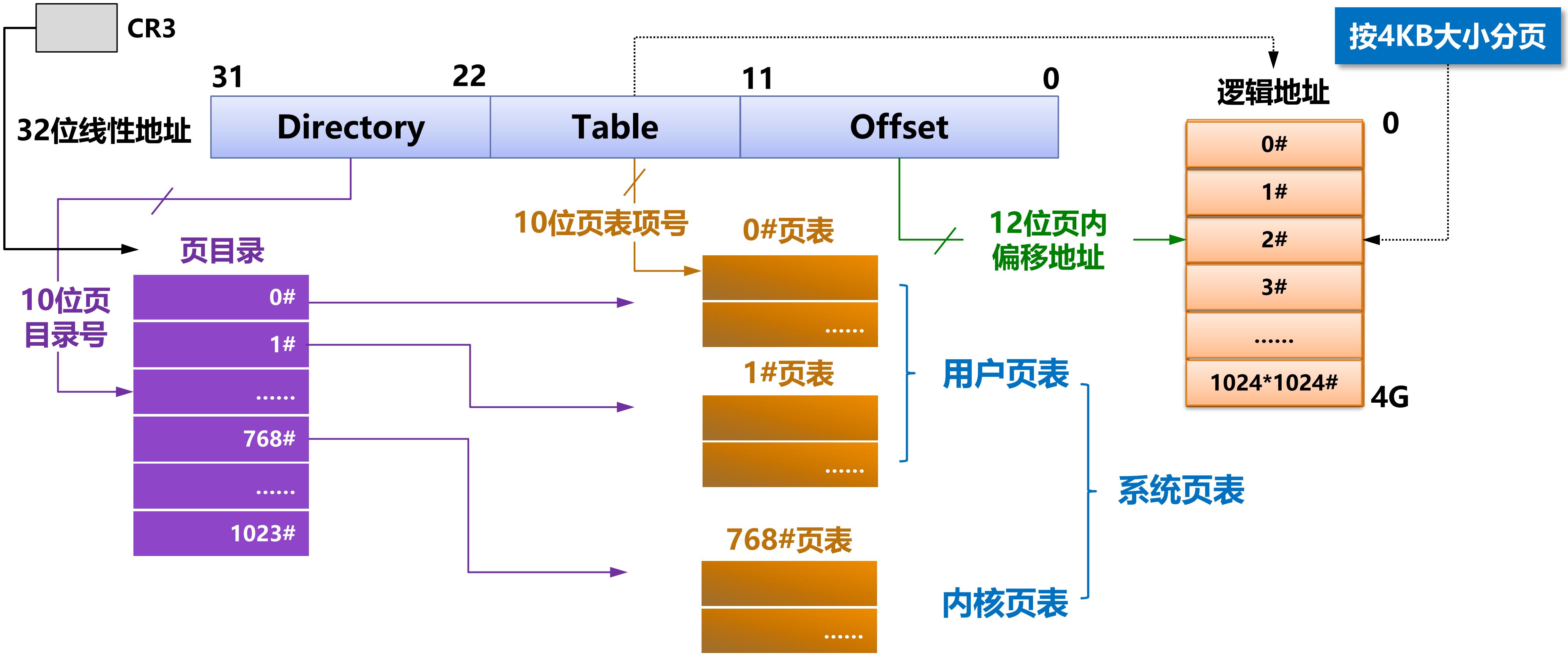
	31 … 22	21 … 12	11 … 0
	Directory	Table	Offset
3G	11 00 0000 00	00 0000 0000	00 00 0000 0000
~	*768#页表*	*1024个页*	
3G+4M-1	11 00 0000 00	11 1111 1111	11 11 1111 1111

G　M　K

页目录

	31 … 12		2	1	0
0#	0#页表所在物理页框		S/U	R/W	P
1#	1#页表所在物理页框		S/U	R/W	P
……					
768#	768#页表所在物理页框		S/U	R/W	P

UNIX V6++程序地址空间



存储空间管理

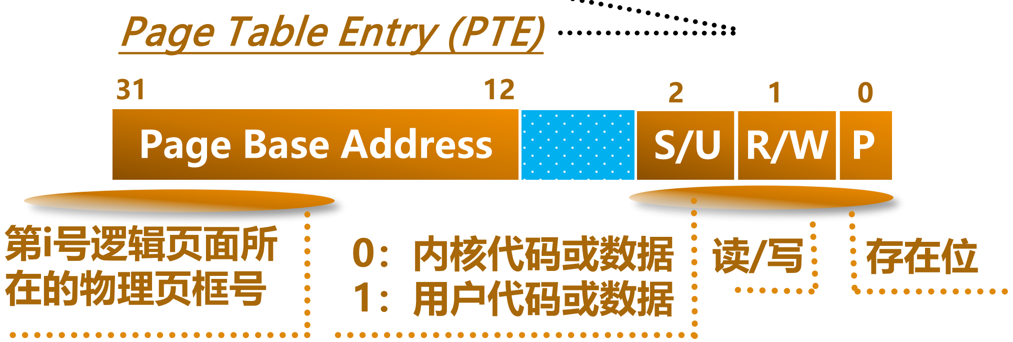
二级页表

```
struct PageDirectoryEntry          /* 页目录中的每一个表项 */
{
    unsigned char m_Present : 1;
    unsigned char m_ReadWriter : 1;
    unsigned char m_UserSupervisor : 1;
    unsigned char m_WriteThrough : 1;
    unsigned char m_CacheDisabled : 1;
    unsigned char m_Accessed : 1;
    unsigned char m_Reserved : 1;
    unsigned char m_PageSize : 1;
    unsigned char m_GlobalPage : 1;
    unsigned char m_ForSystemUser : 3;
    unsigned int m_PageTableBaseAddress : 20;
}__attribute__((packed));

class PageDirectory
{
public:
    ......;
    PageTable& GetPageTableByIdx(int idx);
    PageDirectoryEntry& GetPageDirectoryEntryByIdx(int idx);

public:
    PageDirectoryEntry m_Entrys[1024];
};
```

存储空间管理

二级页表

```
struct PageTableEntry                    /* 页目录中的每一个表项 */
{
    unsigned char m_Present : 1;
    unsigned char m_ReadWriter : 1;
    unsigned char m_UserSupervisor : 1;
    unsigned char m_WriteThrough : 1;
    unsigned char m_CacheDisabled : 1;
    unsigned char m_Accessed : 1;
    unsigned char m_Dirty : 1;
    unsigned char m_PageTableAttribueIndex : 1;
    unsigned char m_GlobalPage : 1;
    unsigned char m_ForSystemUser : 3;
    unsigned int m_PageBaseAddress : 20;
}__attribute__((packed));

class PageTable
{
public:
    static const unsigned int ENTRY_CNT_PER_PAGETABLE = 1024;      /* 每张页表有1024个表项 */
    static const unsigned int SIZE_PER_PAGETABLE_MAP = 0x400000; /* 每张页表可管理4M地址 */
......;
public:
    PageTableEntry m_Entrys[ENTRY_CNT_PER_PAGETABLE];
};
```



主 要 内 容

3.1 存储管理的主要任务

3.2 连续分配方式

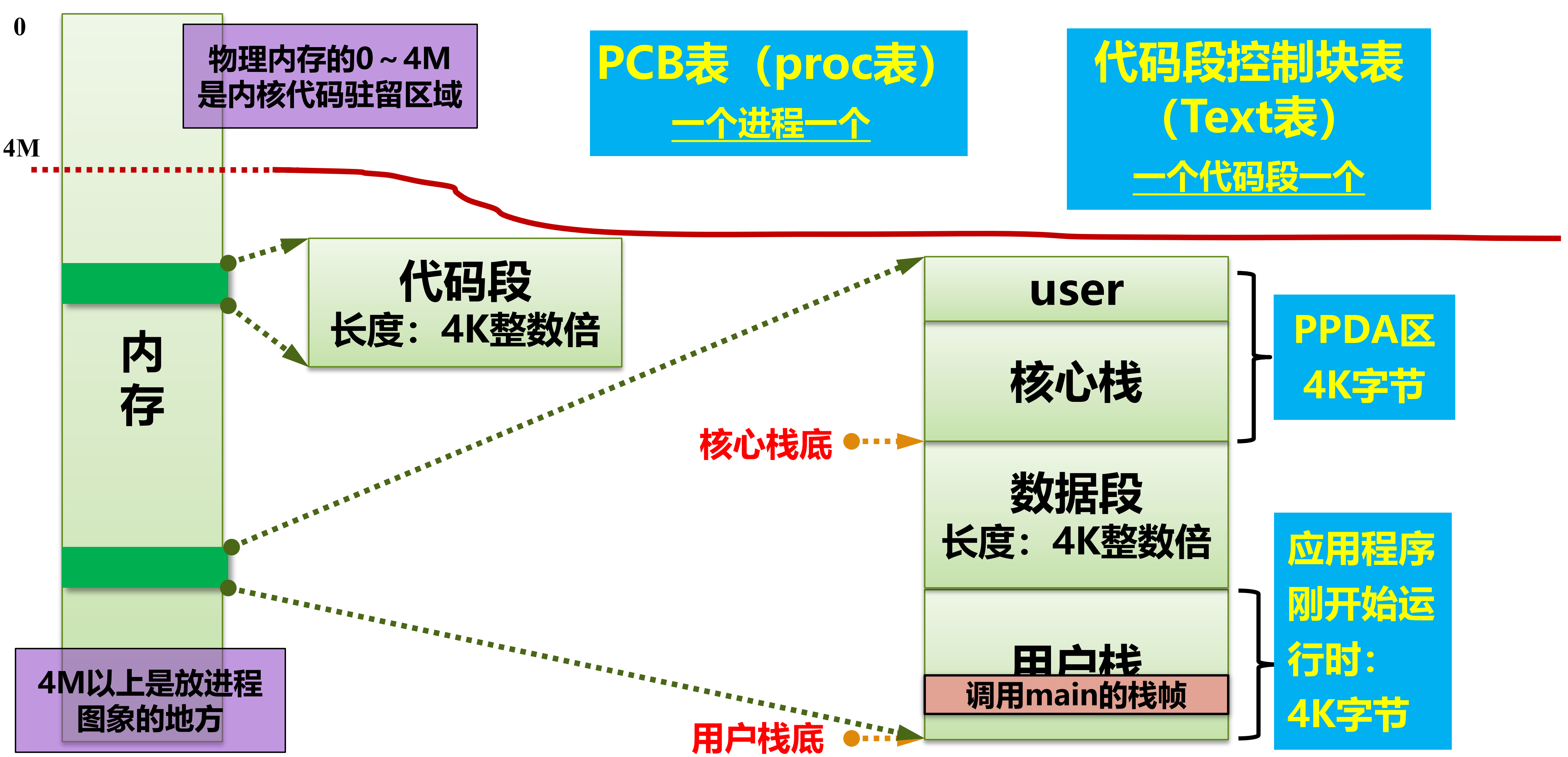
3.3 页式存储管理

3.4 段式与段页式存储管理**

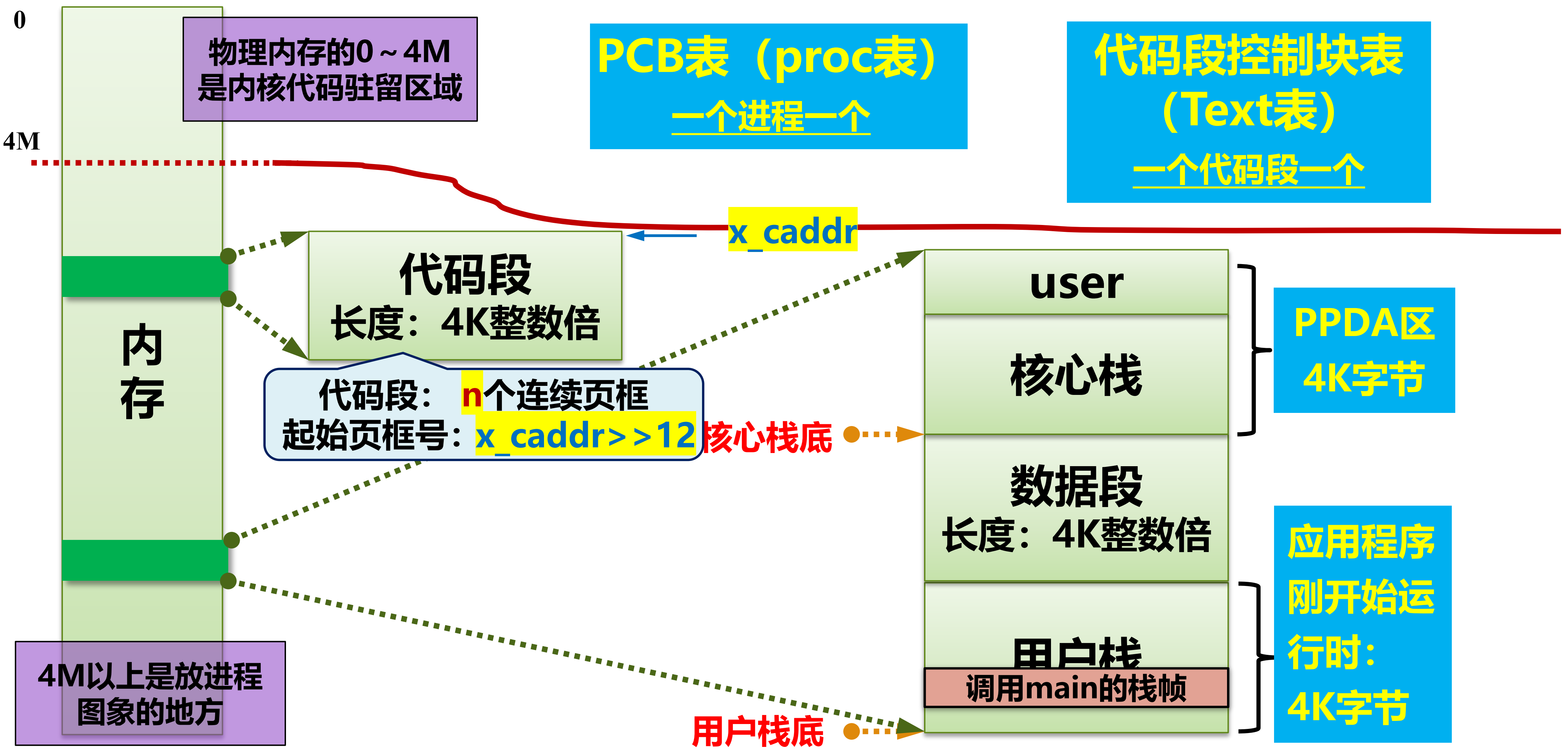
3.5 UNIX 存储管理

- 程序地址空间
- 物理地址空间
- 地址变换
- 存储空间管理

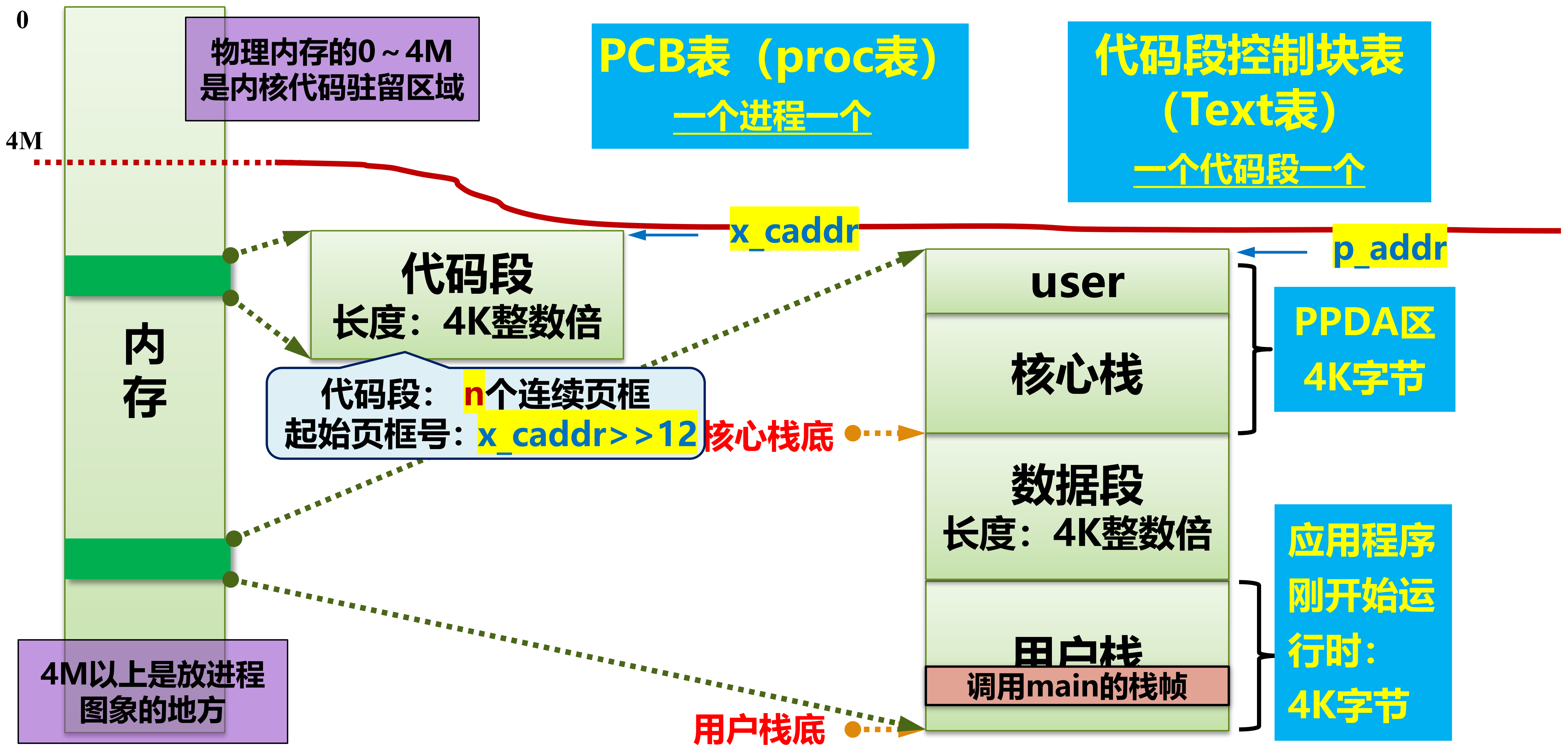
UNIX V6++物理地址空间



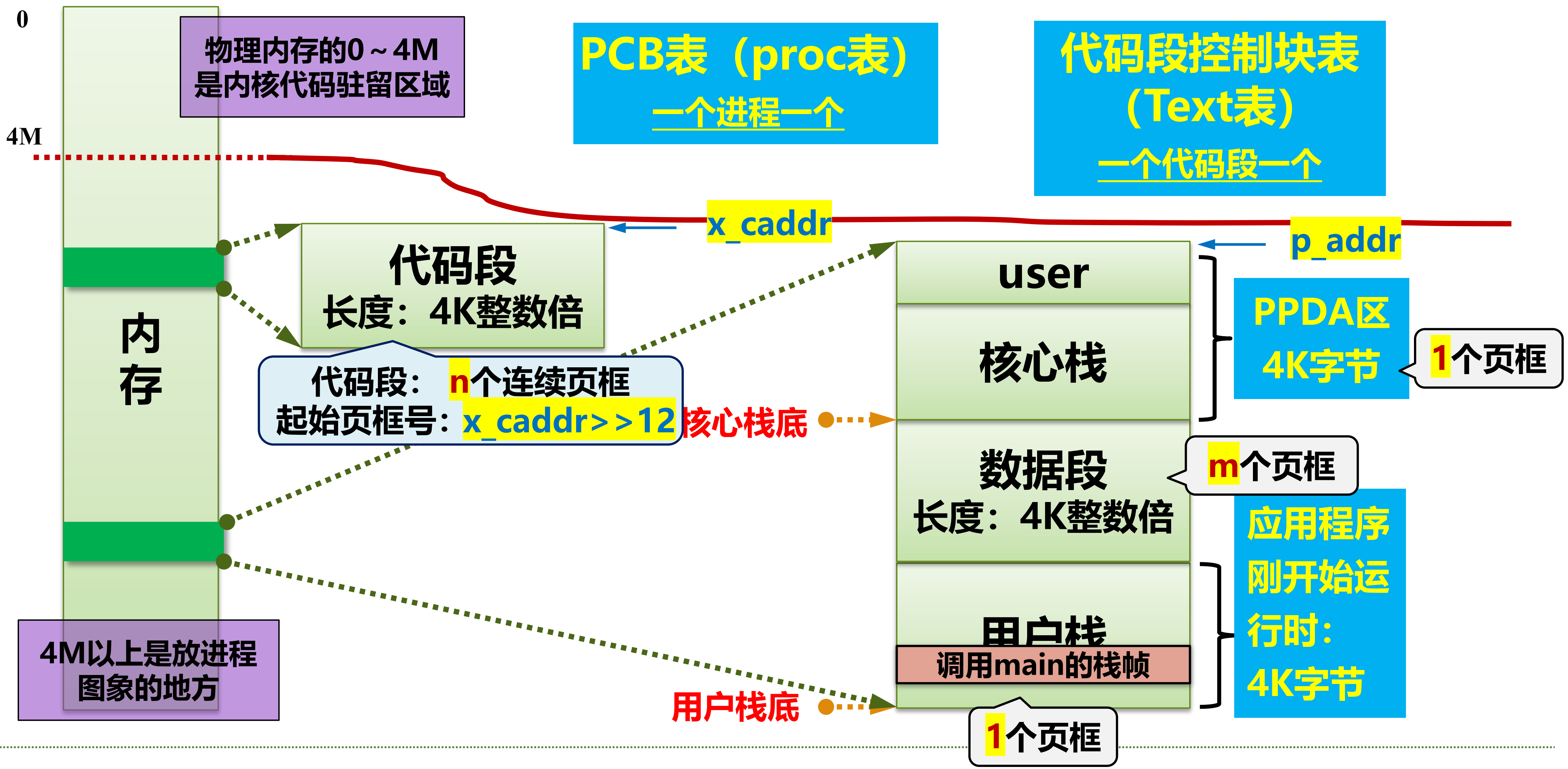
UNIX V6++物理地址空间



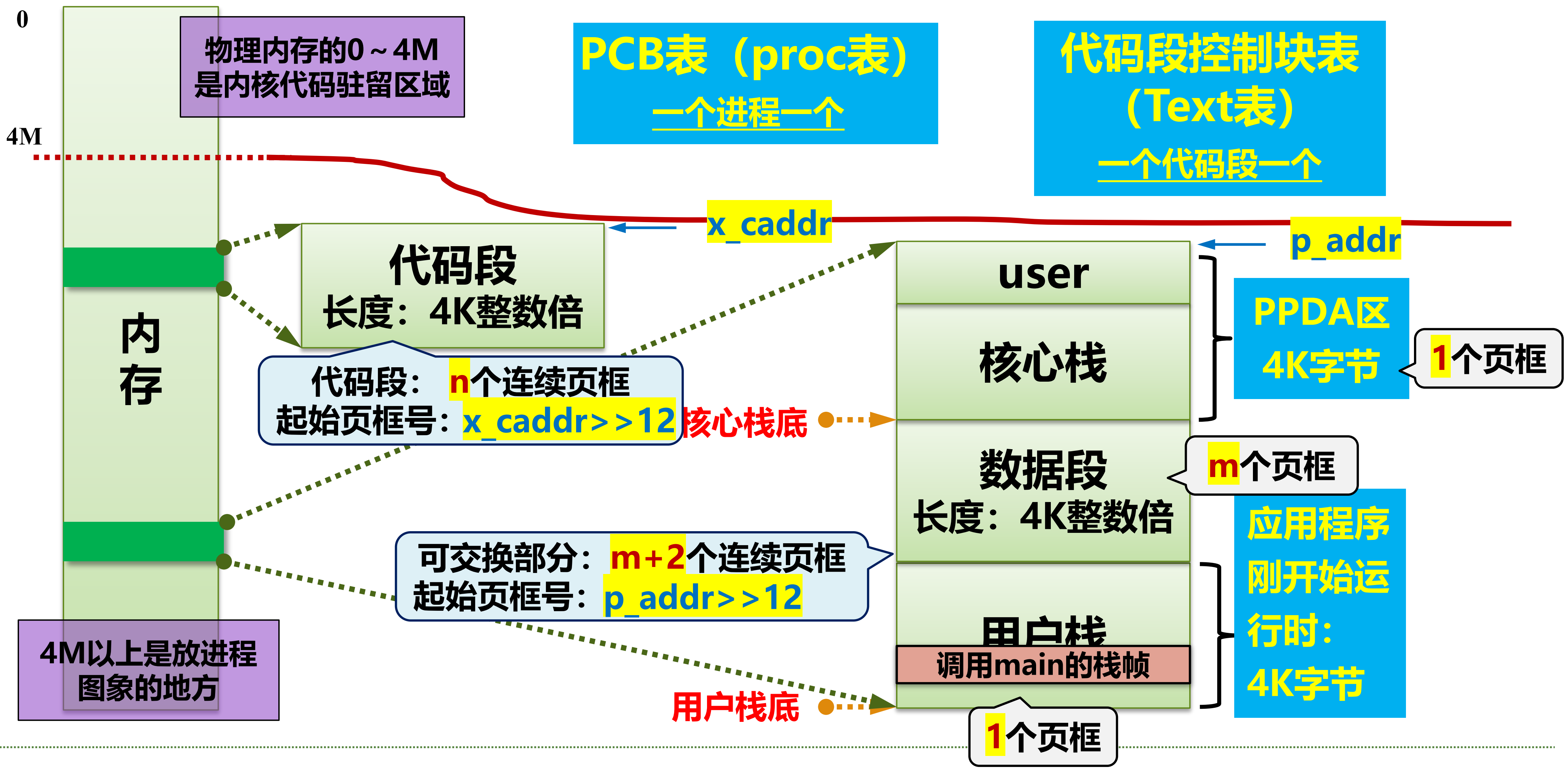
UNIX V6++物理地址空间



UNIX V6++物理地址空间



UNIX V6++物理地址空间



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

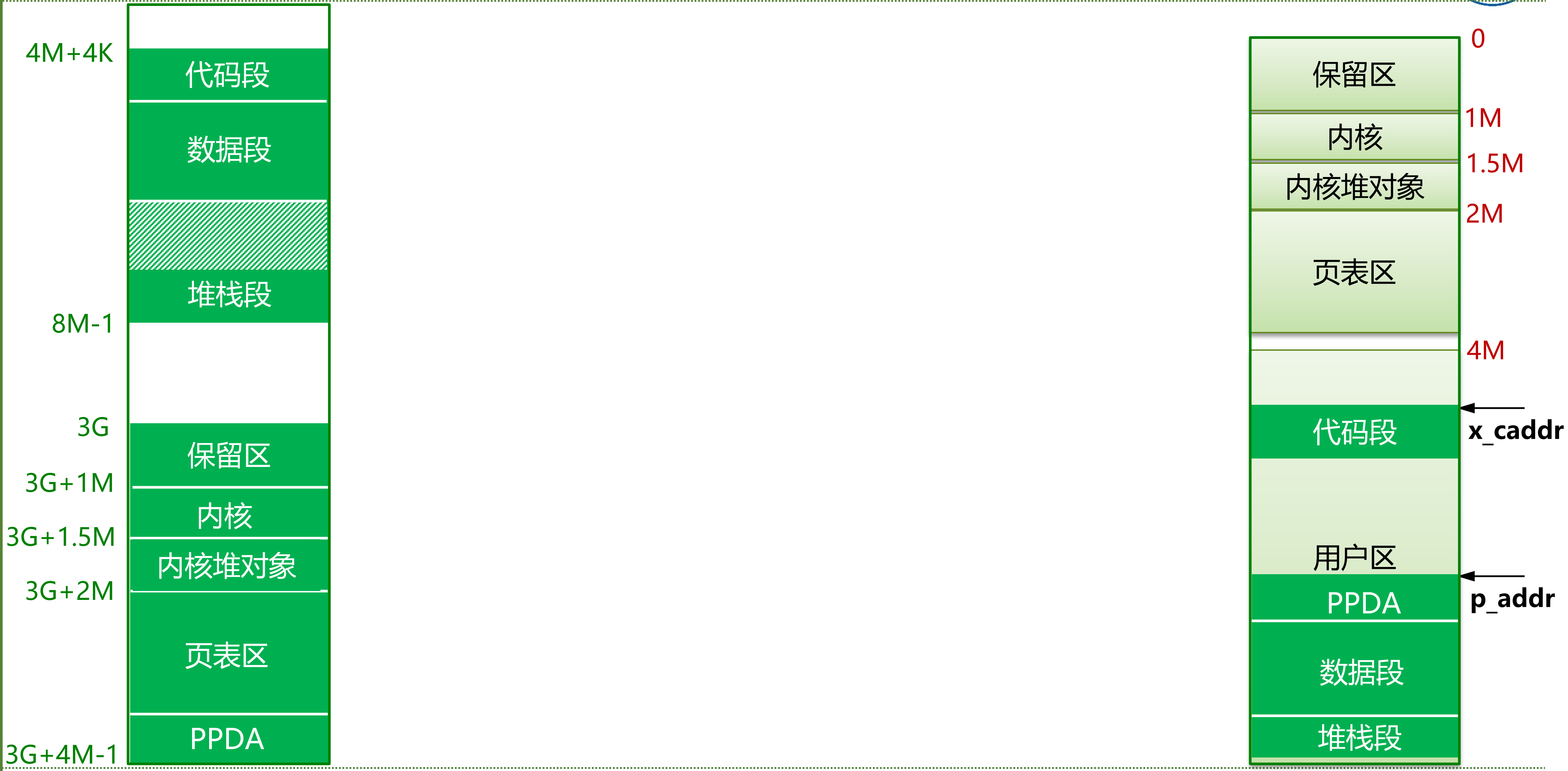
3.4 段式与段页式存储管理**

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

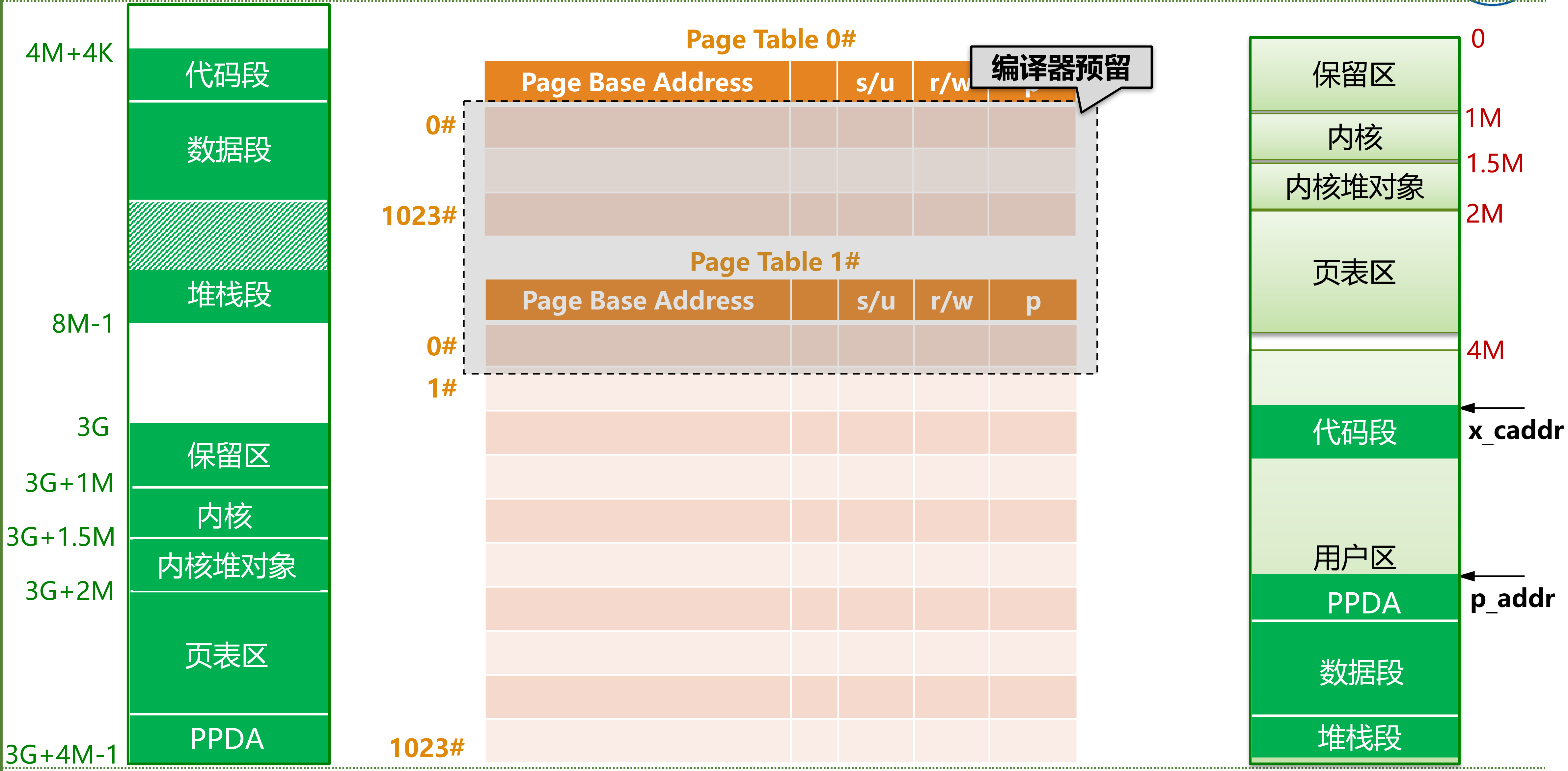
UNIX V6++的地址变换

页表的构成



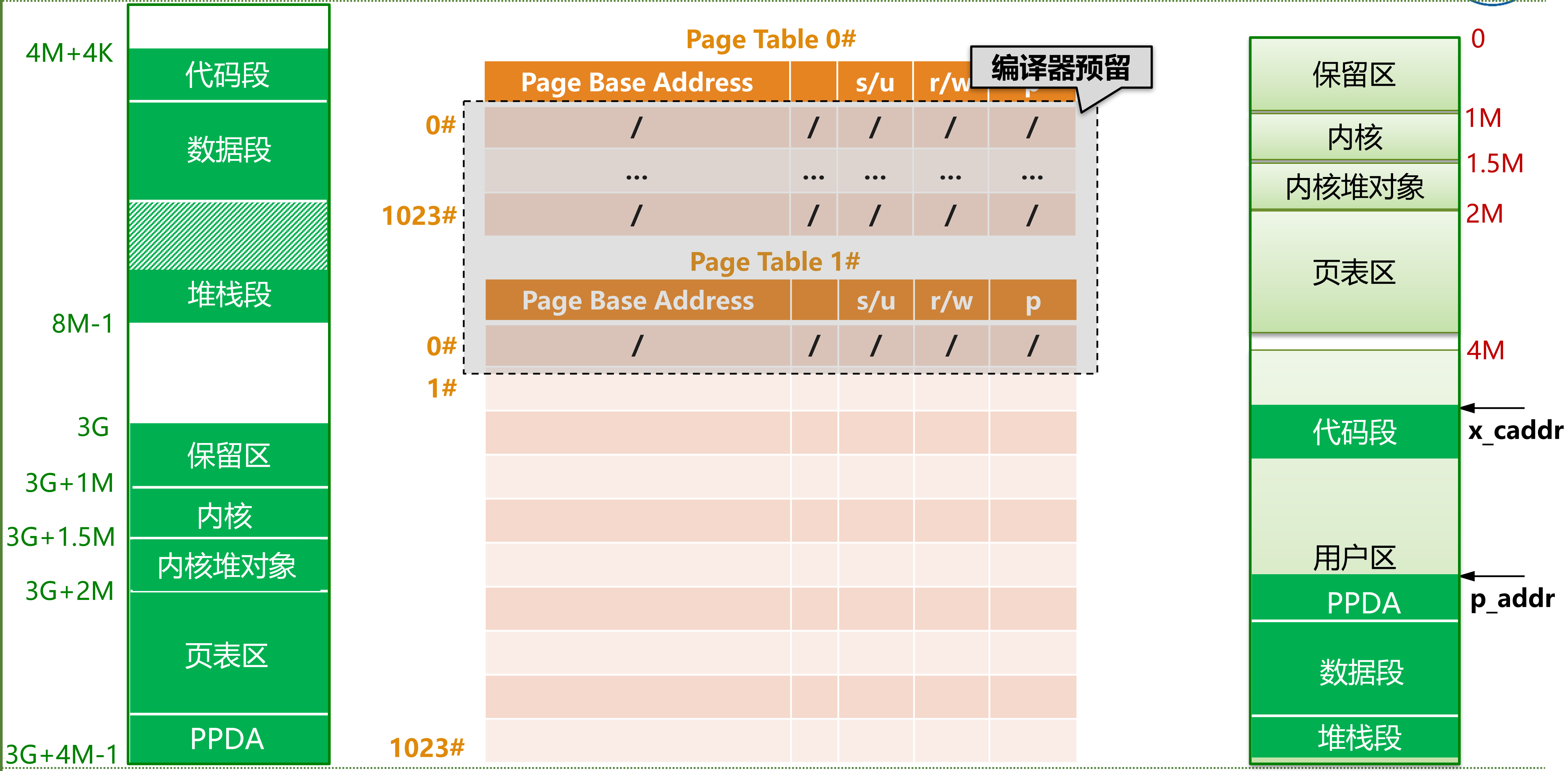
UNIX V6++的地址变换

页表的构成



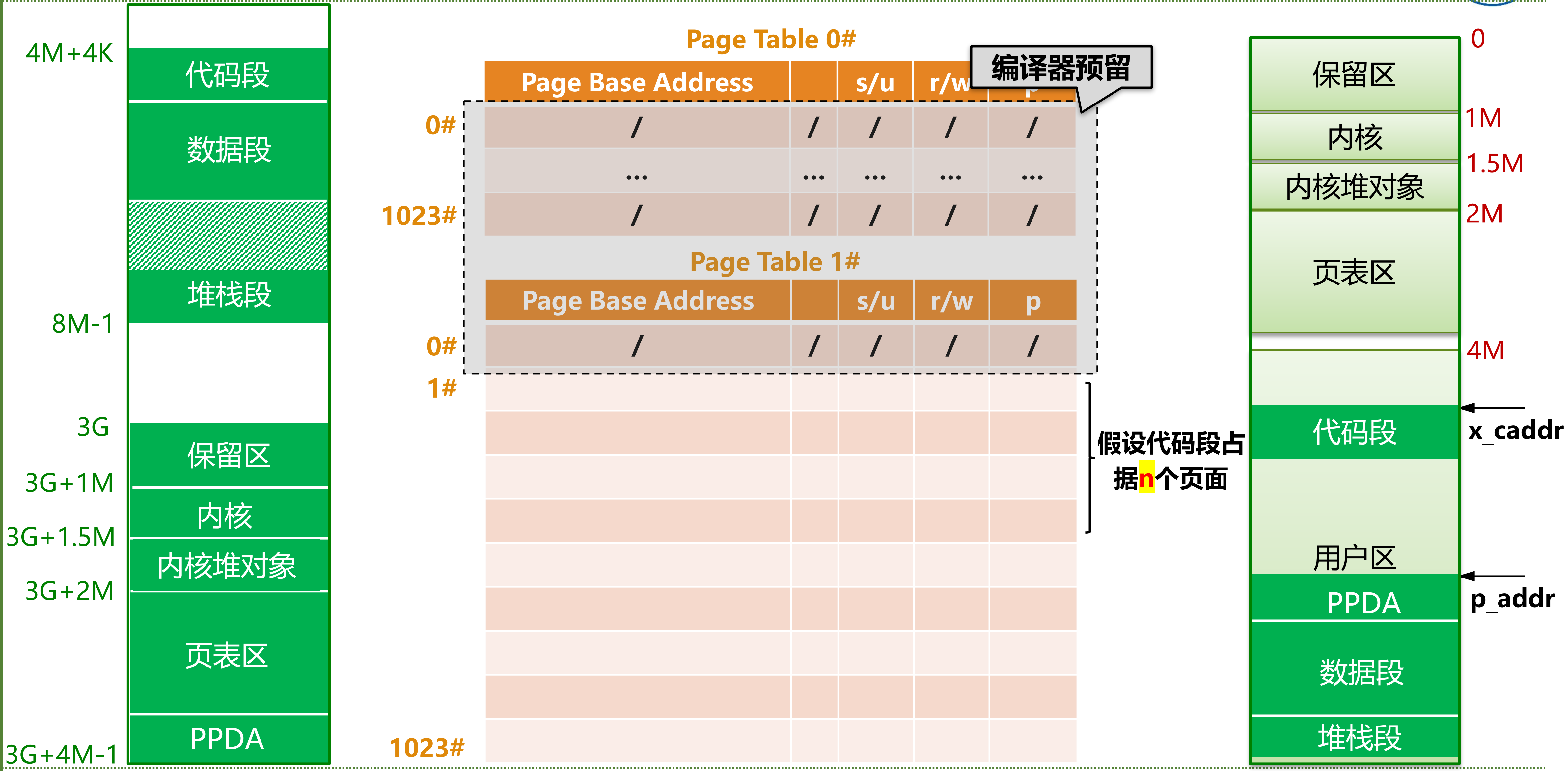
UNIX V6++的地址变换

页表的构成



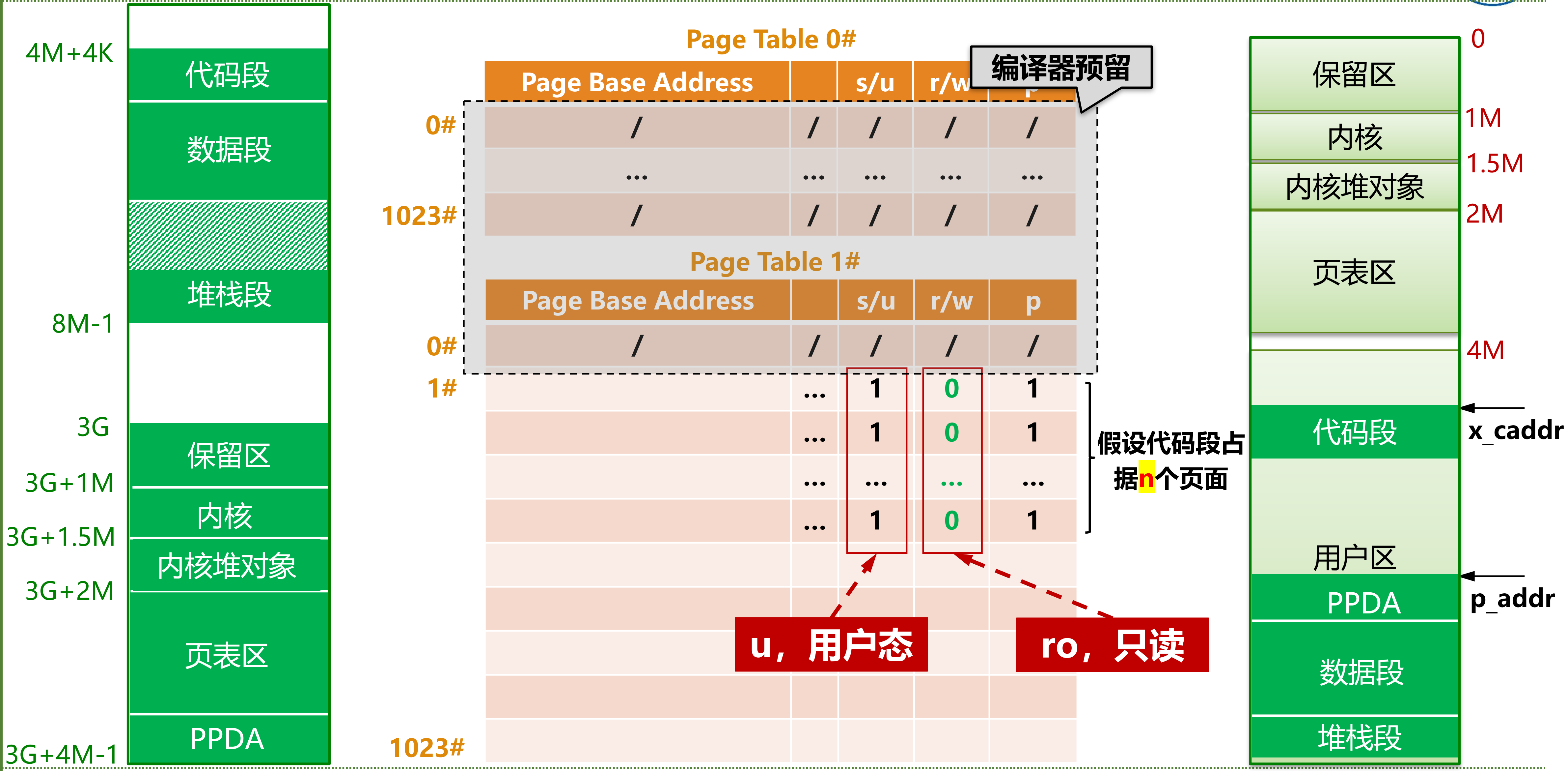
UNIX V6++的地址变换

页表的构成



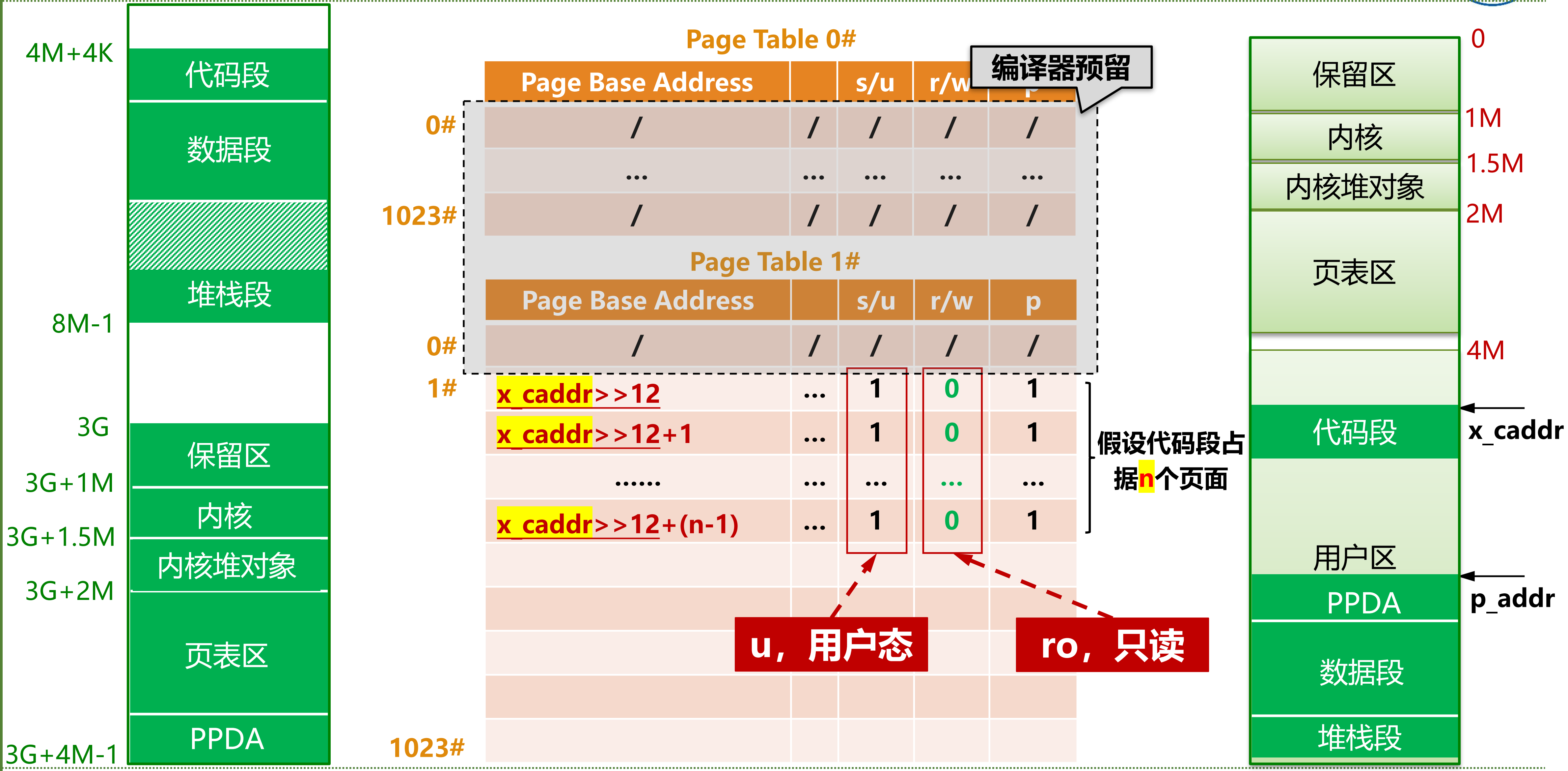
UNIX V6++的地址变换

页表的构成



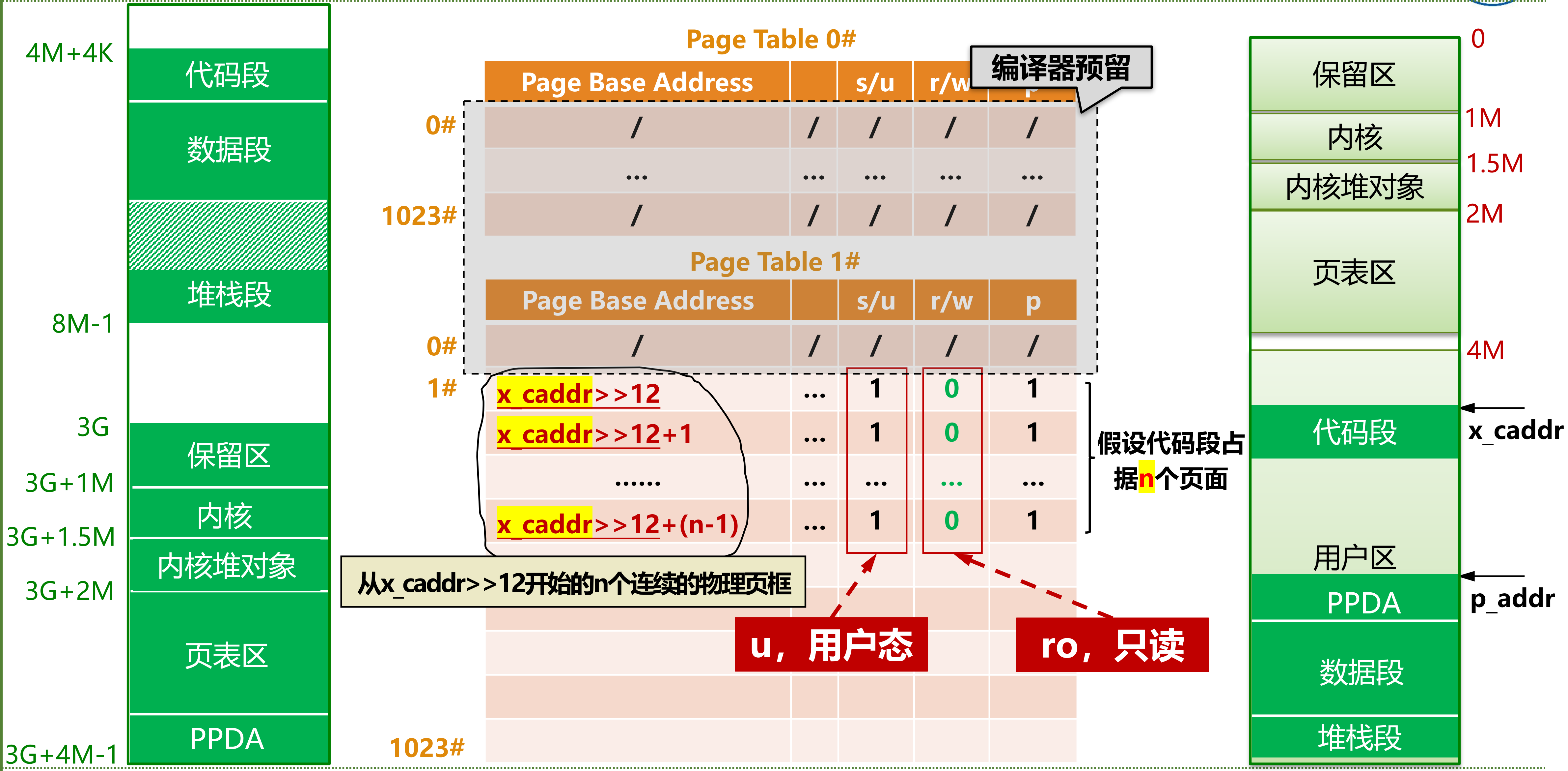
UNIX V6++的地址变换

页表的构成



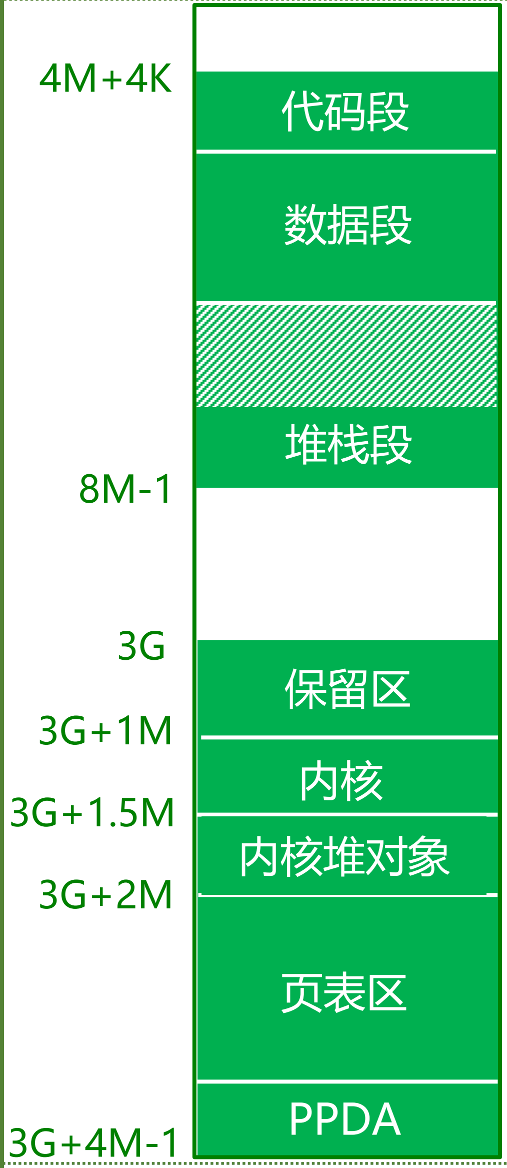
UNIX V6++的地址变换

页表的构成



UNIX V6++的地址变换

页表的构成



Page Table 0#

编译器预留

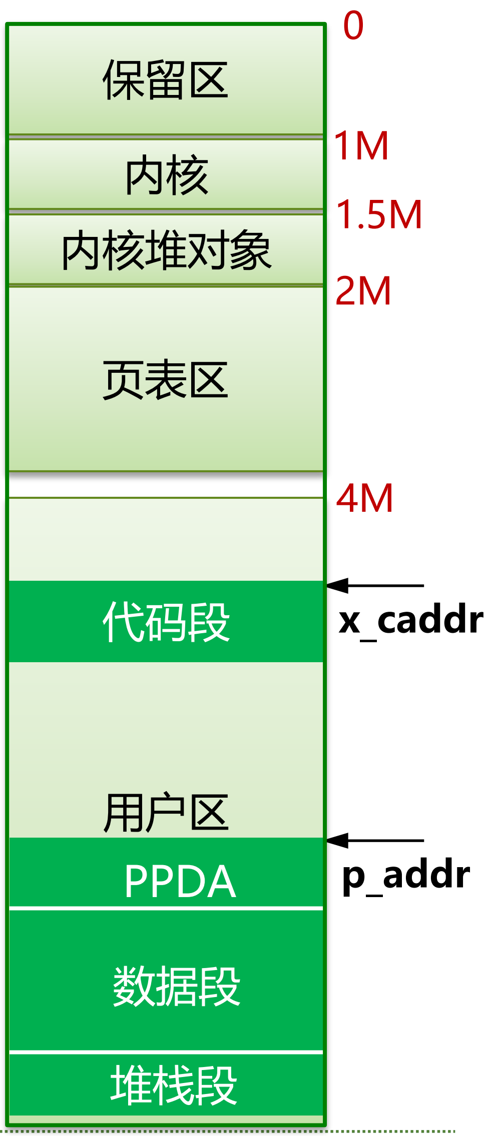
	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
	…	…	…	…	…
1023#	/	/	/	/	/

Page Table 1#

	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12	…	1	0	1
	x_caddr>>12+1	…	1	0	1
	……	…	…	…	…
	x_caddr>>12+(n-1)	…	1	0	1
1023#					

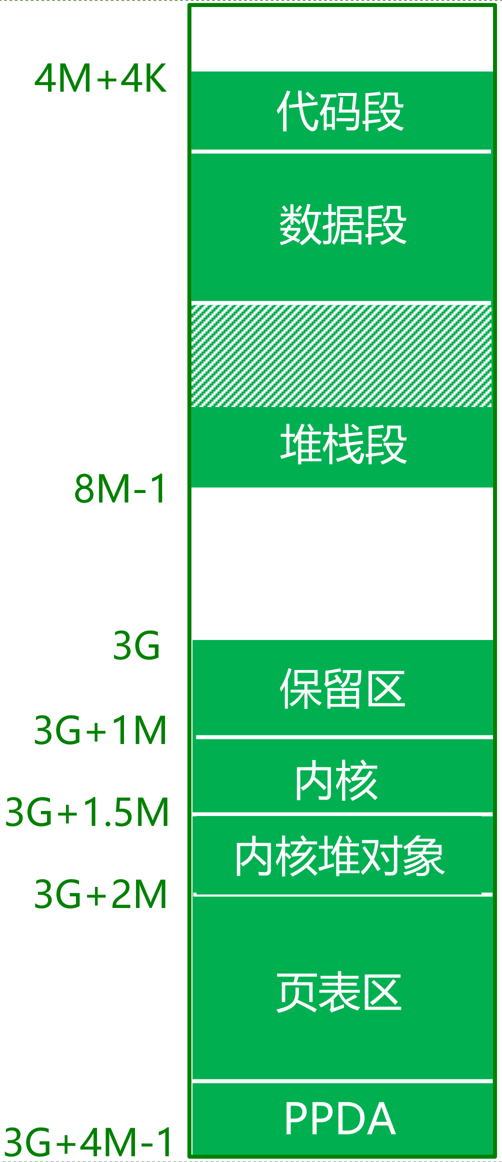
假设代码段占据n个页面

假设数据段占据m个页面



UNIX V6++的地址变换

页表的构成



Page Table 0#

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

编译器预留

Page Table 1#

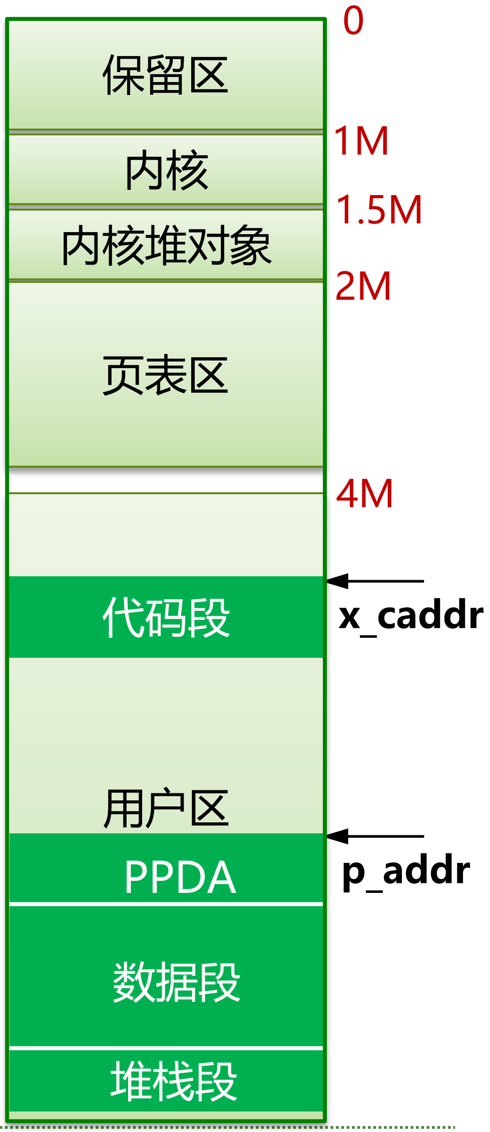
	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12	…	1	0	1
	x_caddr>>12+1	…	1	0	1
	……	…	…	…	…
	x_caddr>>12+(n-1)	…	1	0	1
		…	1	1	1
		…	…	…	…
		…	1	1	1
1023#					

假设代码段占据n个页面

假设数据段占据m个页面

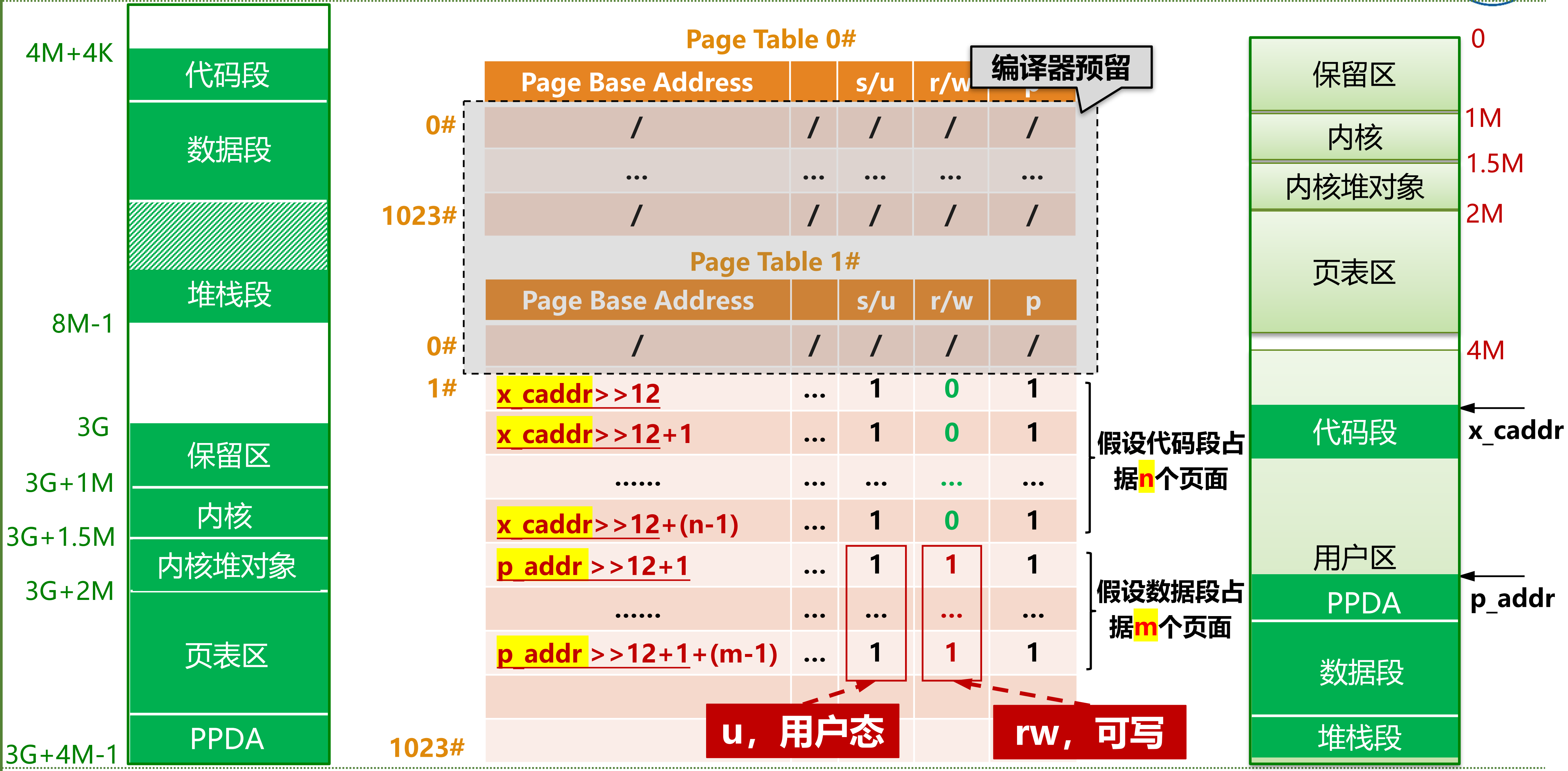
u，用户态

rw，可写



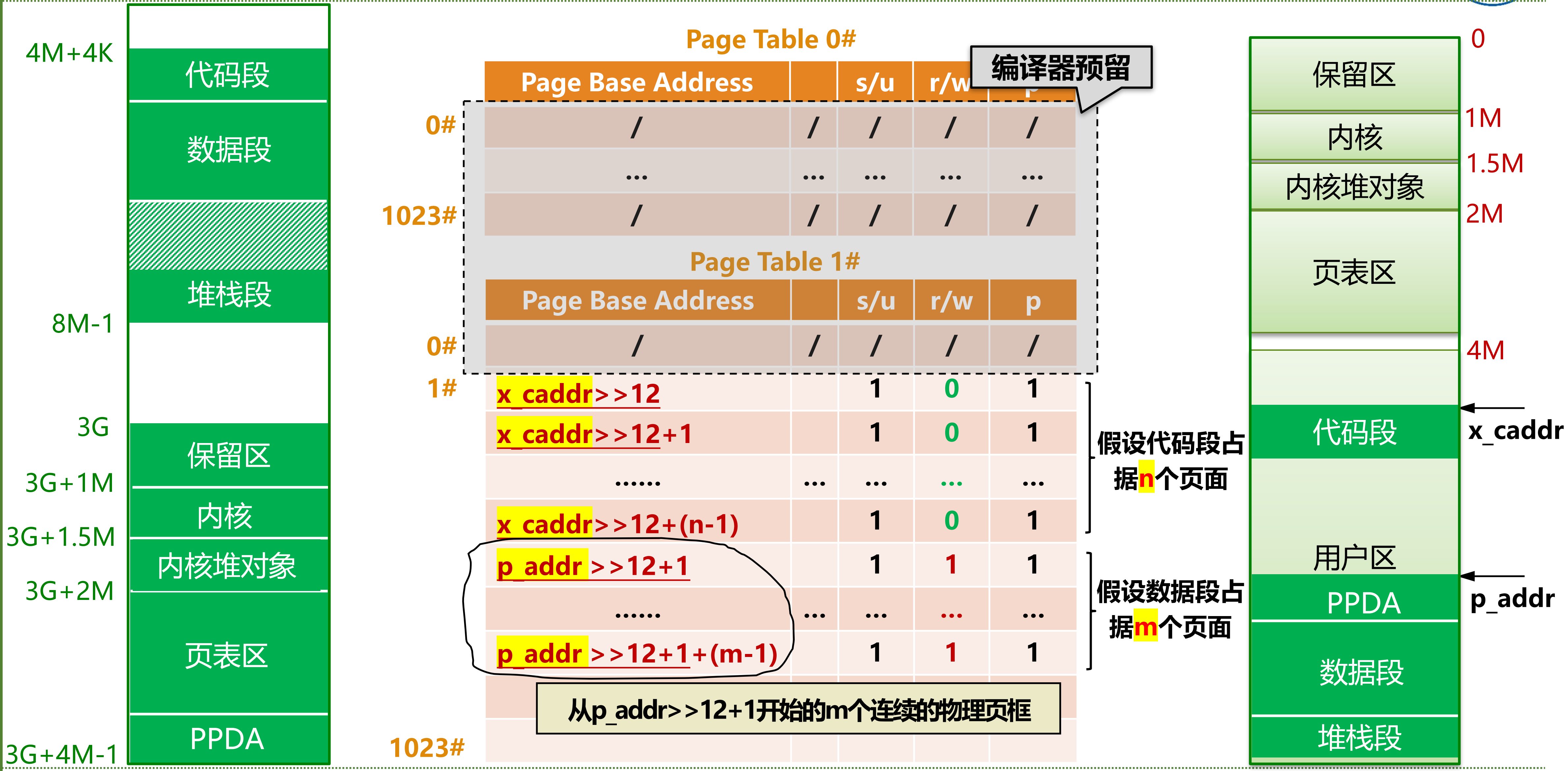
UNIX V6++的地址变换

页表的构成



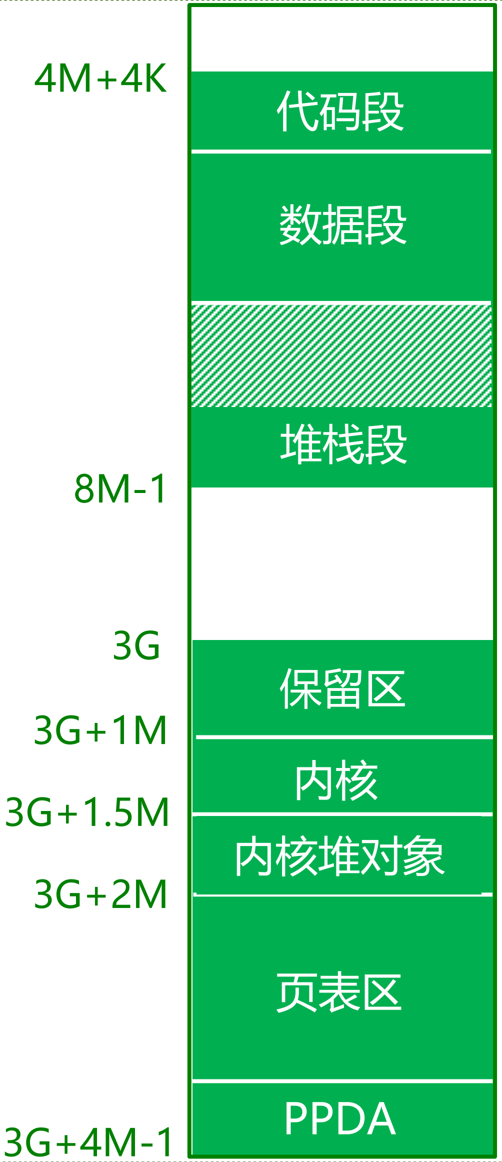
UNIX V6++的地址变换

页表的构成



UNIX V6++的地址变换

页表的构成



Page Table 0#

编译器预留

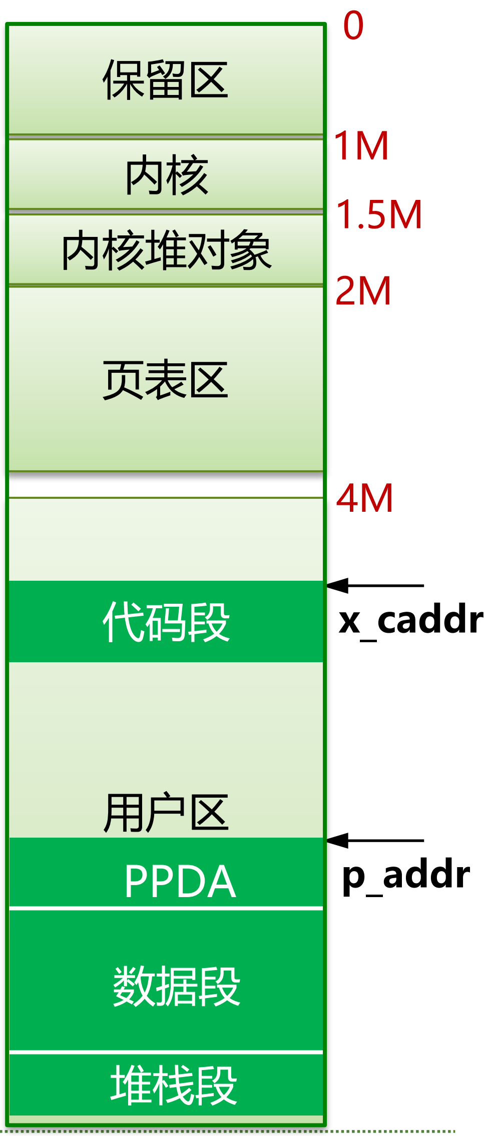
	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
	…	…	…	…	…
1023#	/	/	/	/	/

Page Table 1#

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

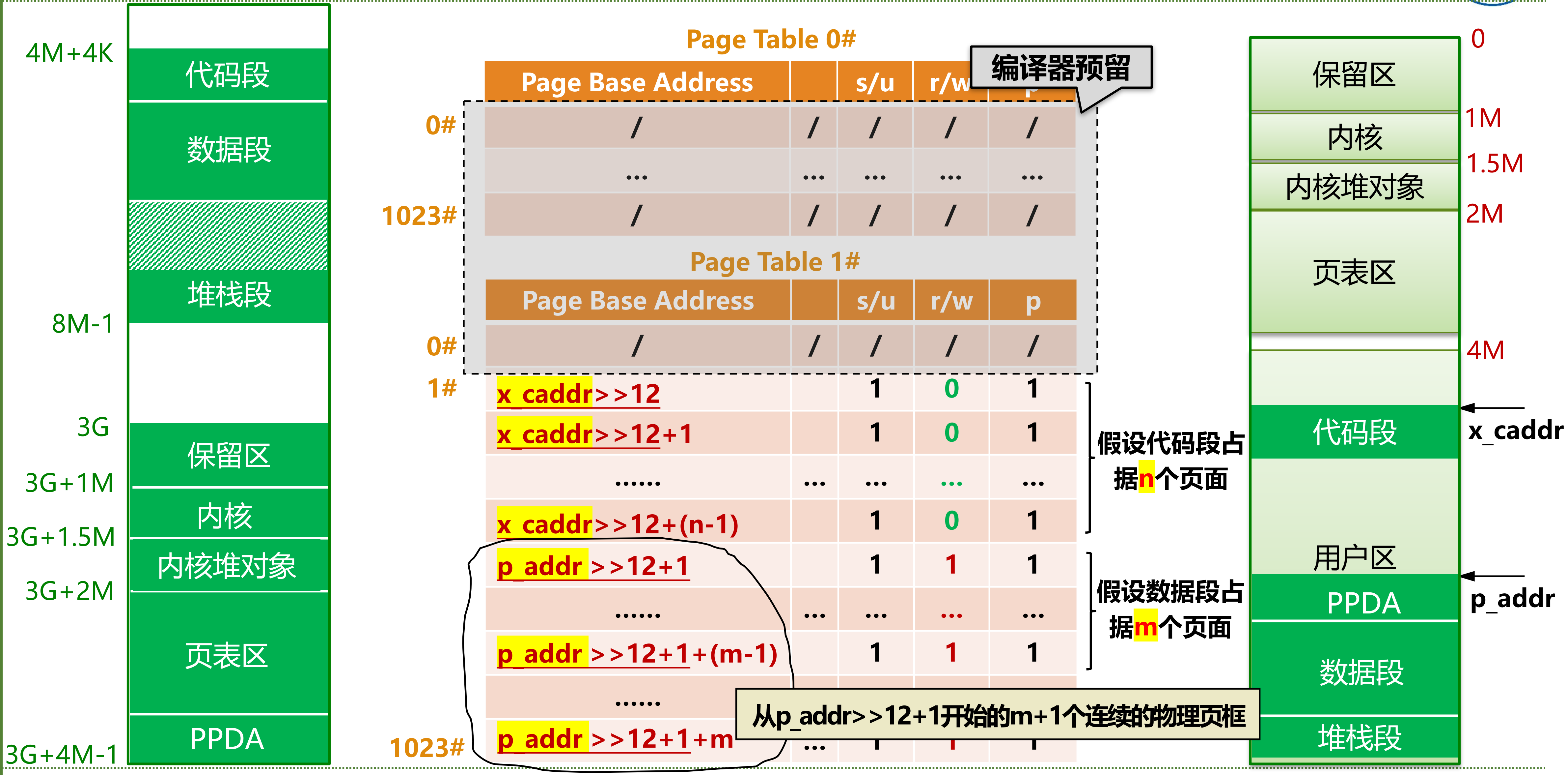
假设代码段占据n个页面

假设数据段占据m个页面



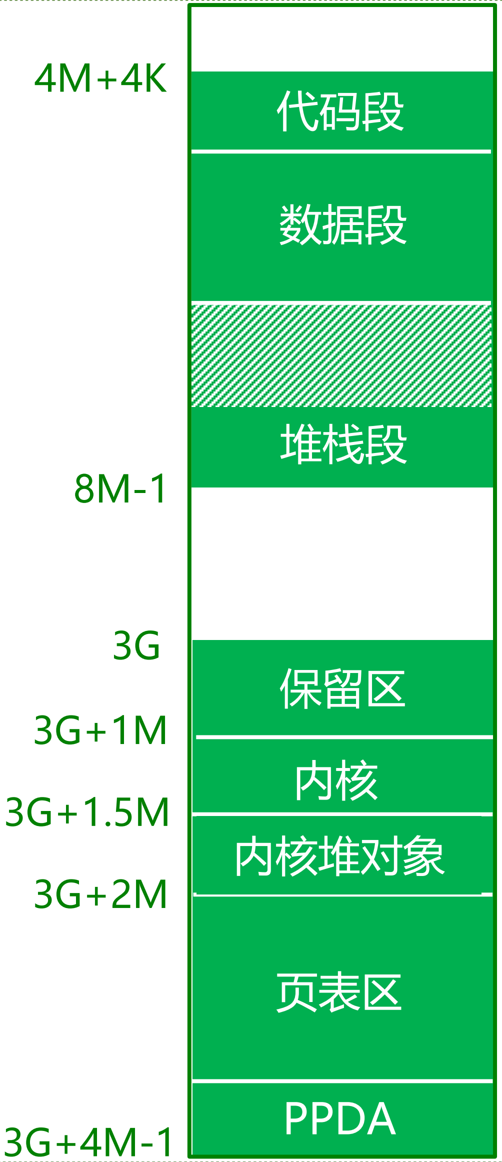
UNIX V6++的地址变换

页表的构成



UNIX V6++的地址变换

页表的构成



Page Table 0#

编译器预留

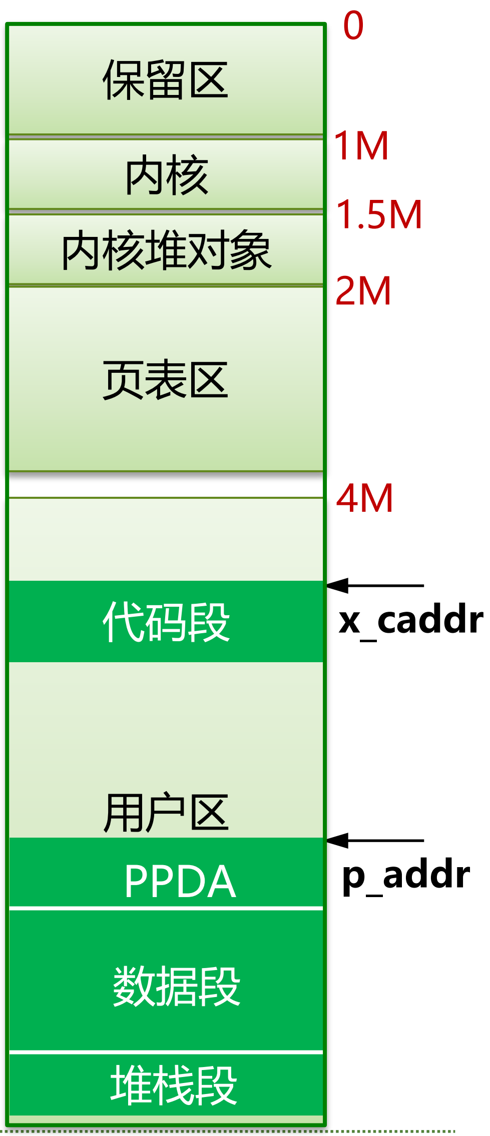
	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
	…	…	…	…	…
1023#	/	/	/	/	/

Page Table 1#

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

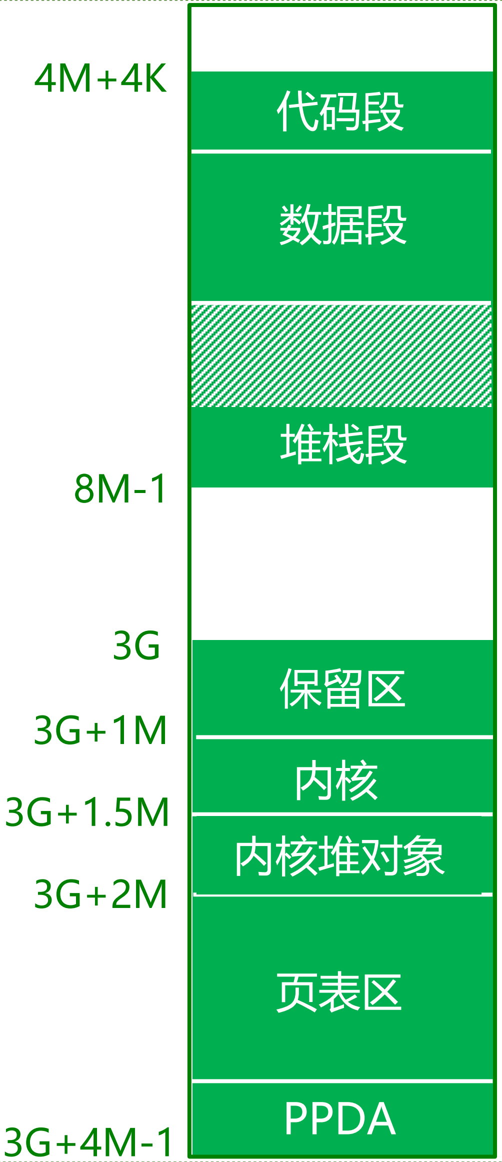
假设代码段占据n个页面

假设数据段占据m个页面



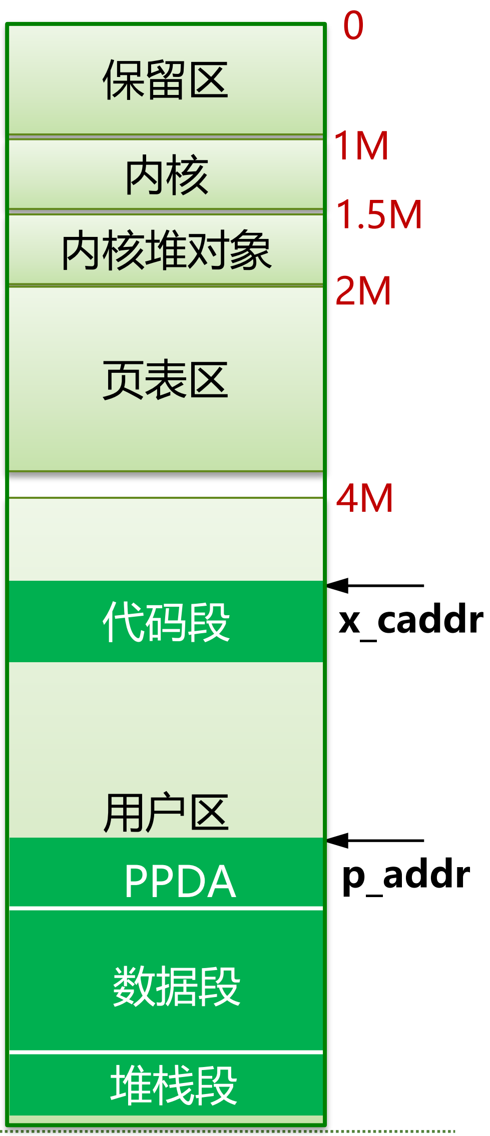
UNIX V6++的地址变换

页表的构成



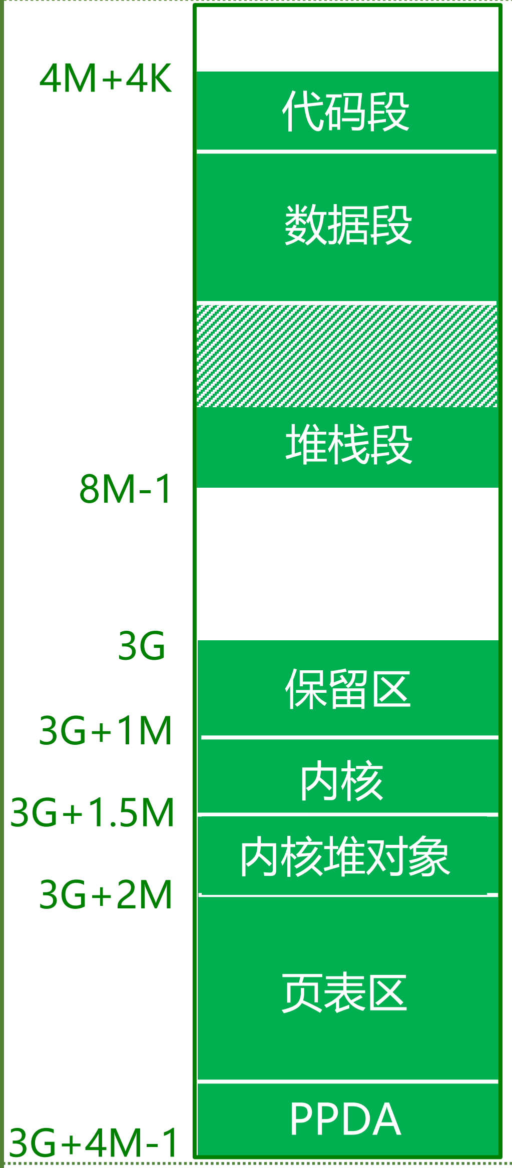
Page Table 768#

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



UNIX V6++的地址变换

页表的构成

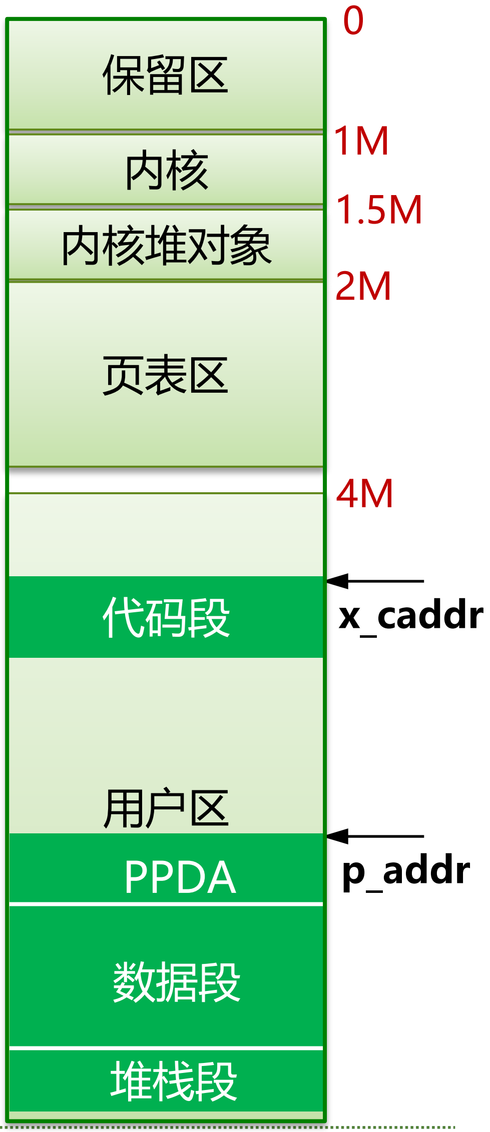


Page Table 768#

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

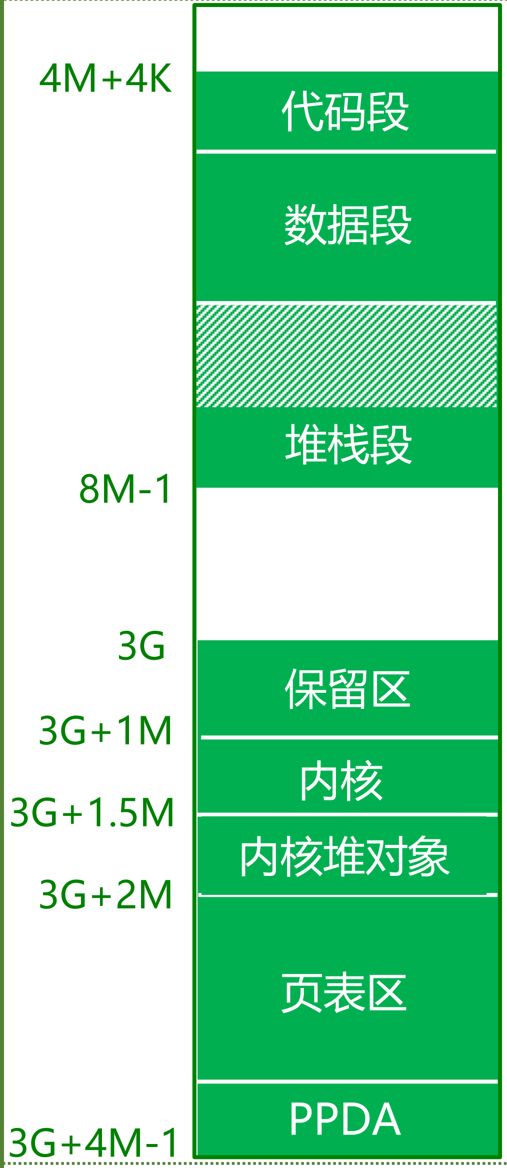
s，内核态

rw，读写



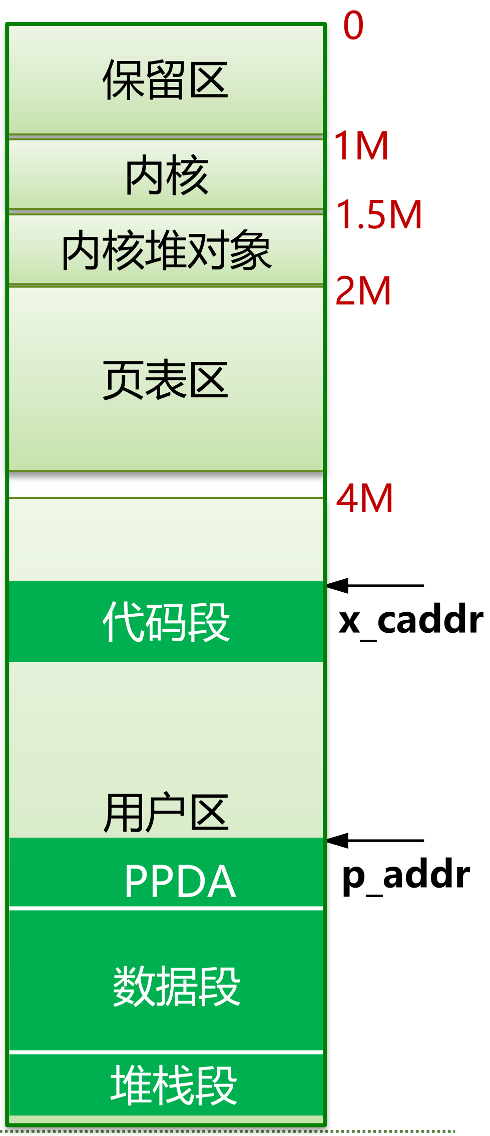
UNIX V6++的地址变换

页表的构成



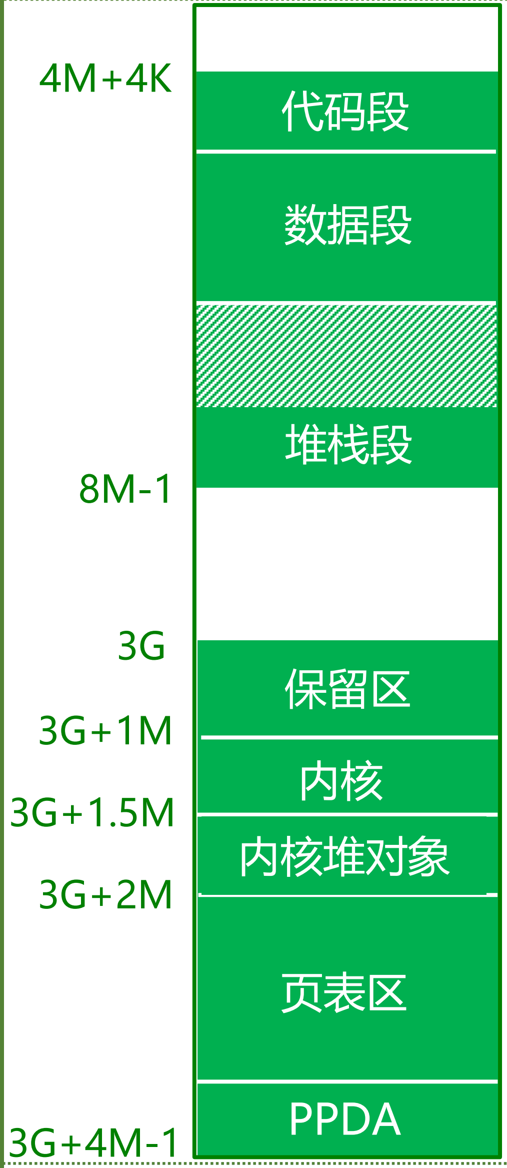
Page Table 768#

	Page Base Address		s/u	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



UNIX V6++的地址变换

页表的构成



Page Table 0#

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

Page Table 1#

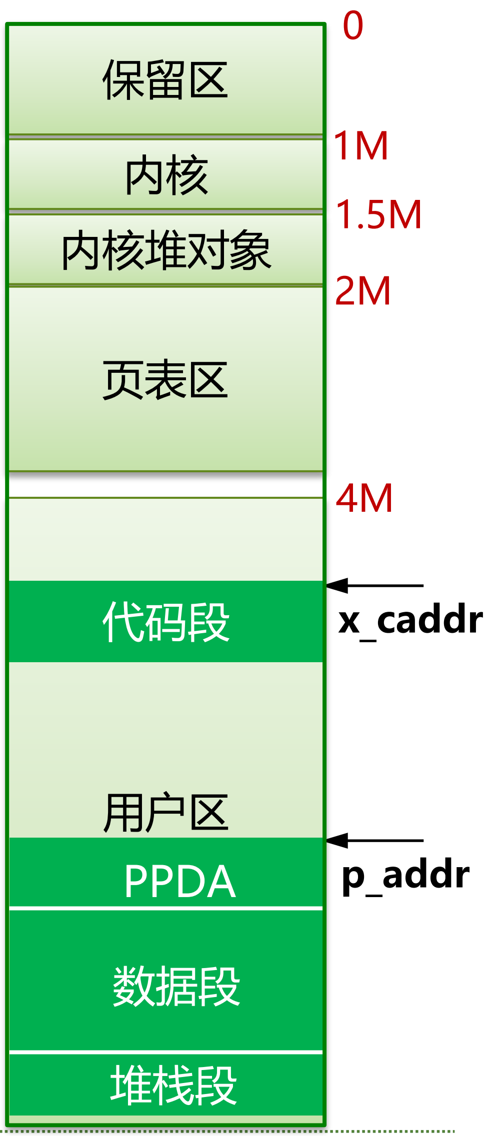
	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
1#	x_caddr>>12		1	0	1
	x_caddr>>12+1		1	0	1
	......	...	...	...	...
	x_caddr>>12+(n-1)		1	0	1
	p_addr >>12+1		1	1	1
	......	...	...	...	...
	p_addr >>12+1+(m-1)		1	1	1
	......	...	...	...	...
1023#	p_addr >>12+1+m	...	1	1	1

Page Table 768#

	Page Base Address		s/u	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

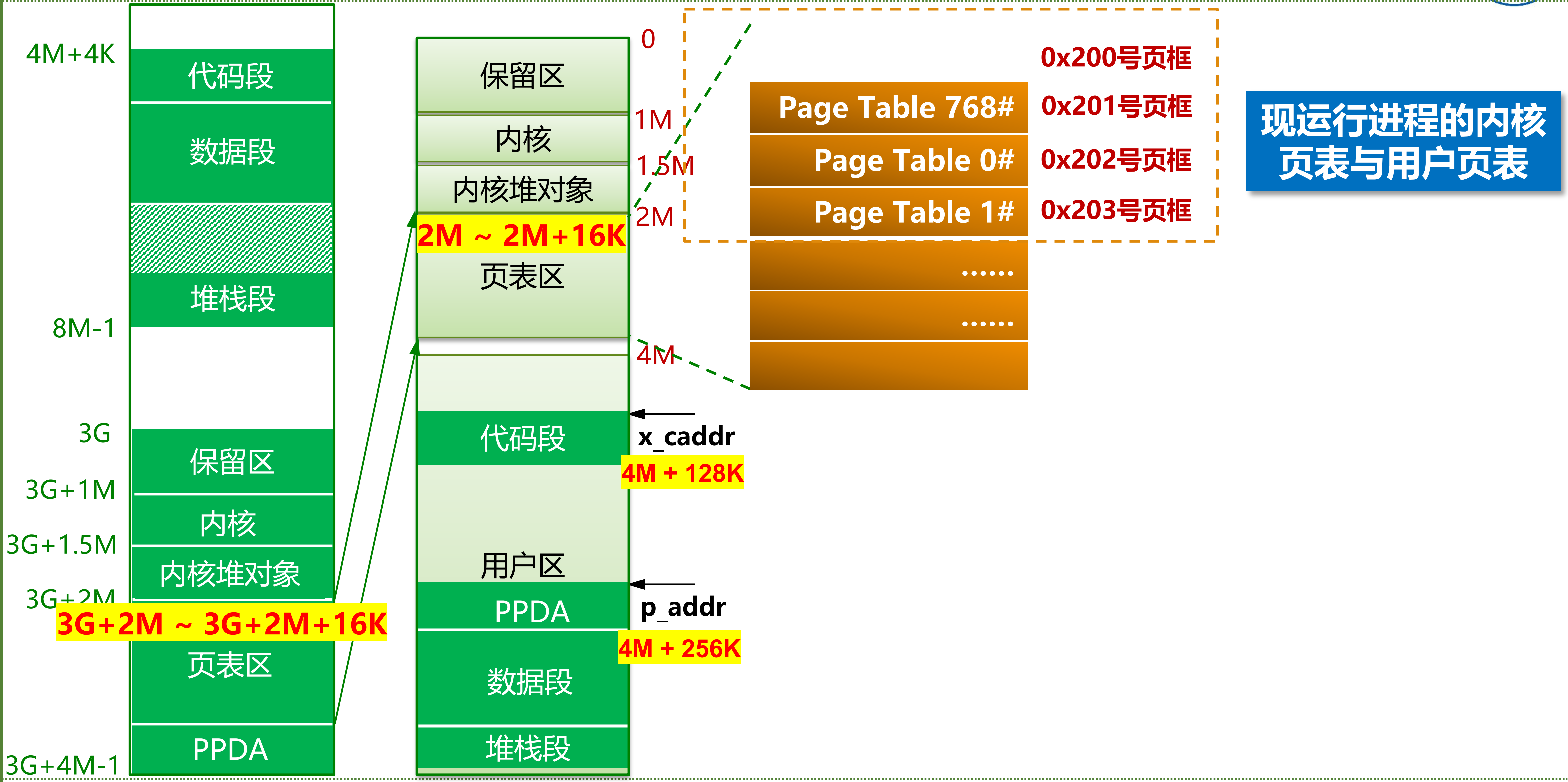
三张页表保存在哪里？

页目录保存在哪里？



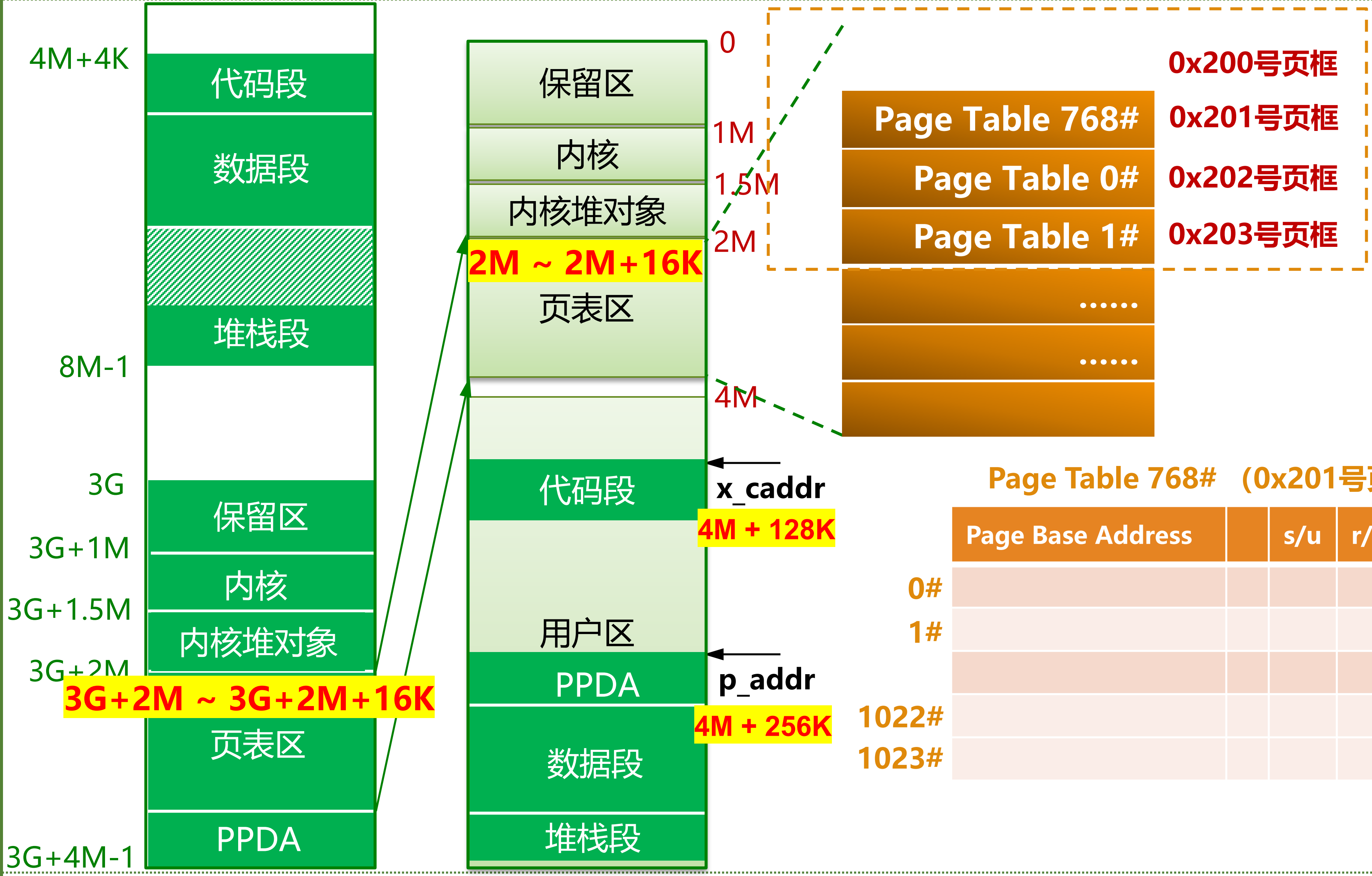
UNIX V6++的地址变换

现运行进程的页表



UNIX V6++的地址变换

现运行进程的页表



现运行进程的内核页表与用户页表

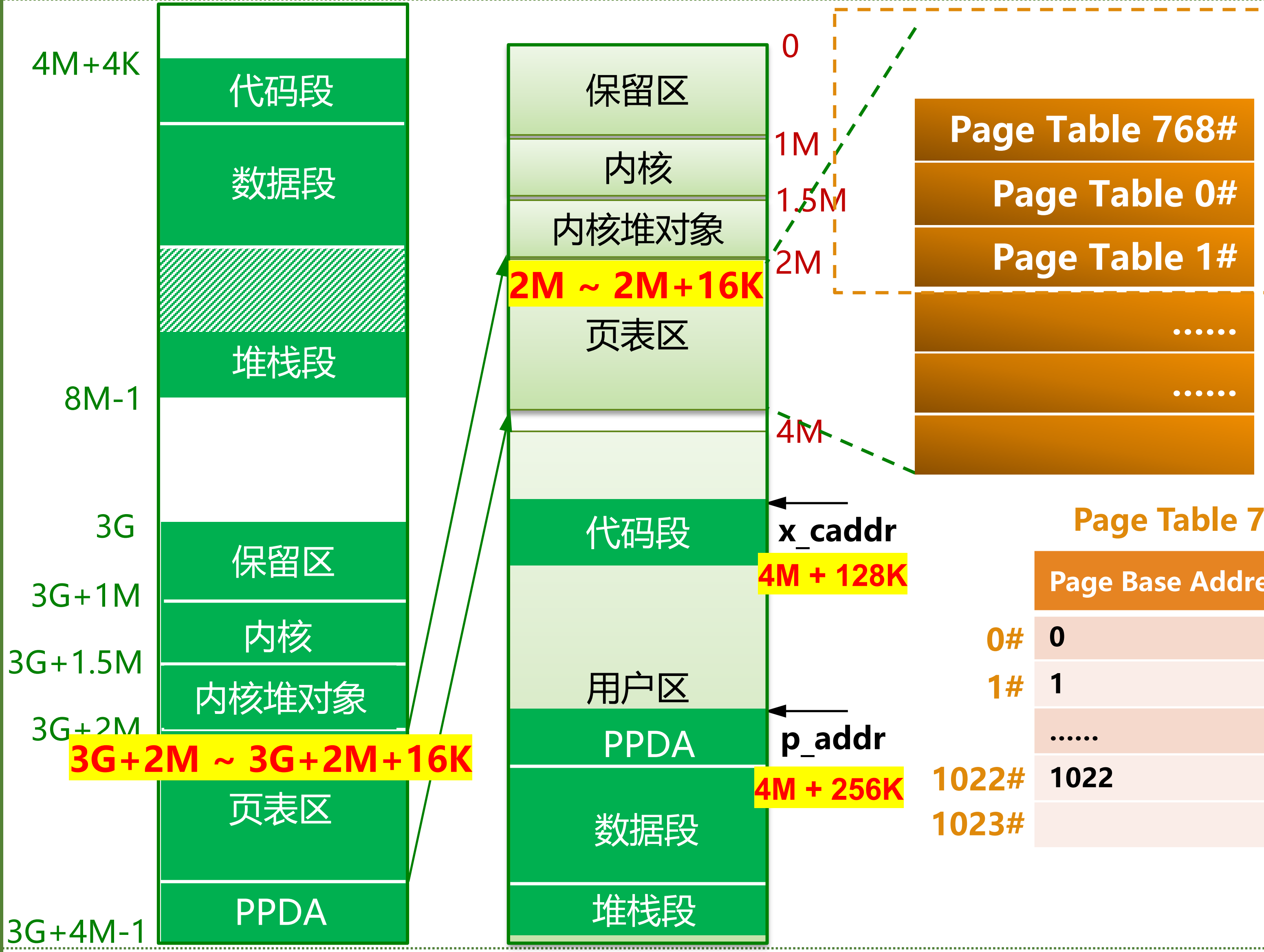
Page Table 768# （0x201号页框）

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

UNIX V6++的地址变换

现运行进程的页表

现运行进程的内核页表与用户页表

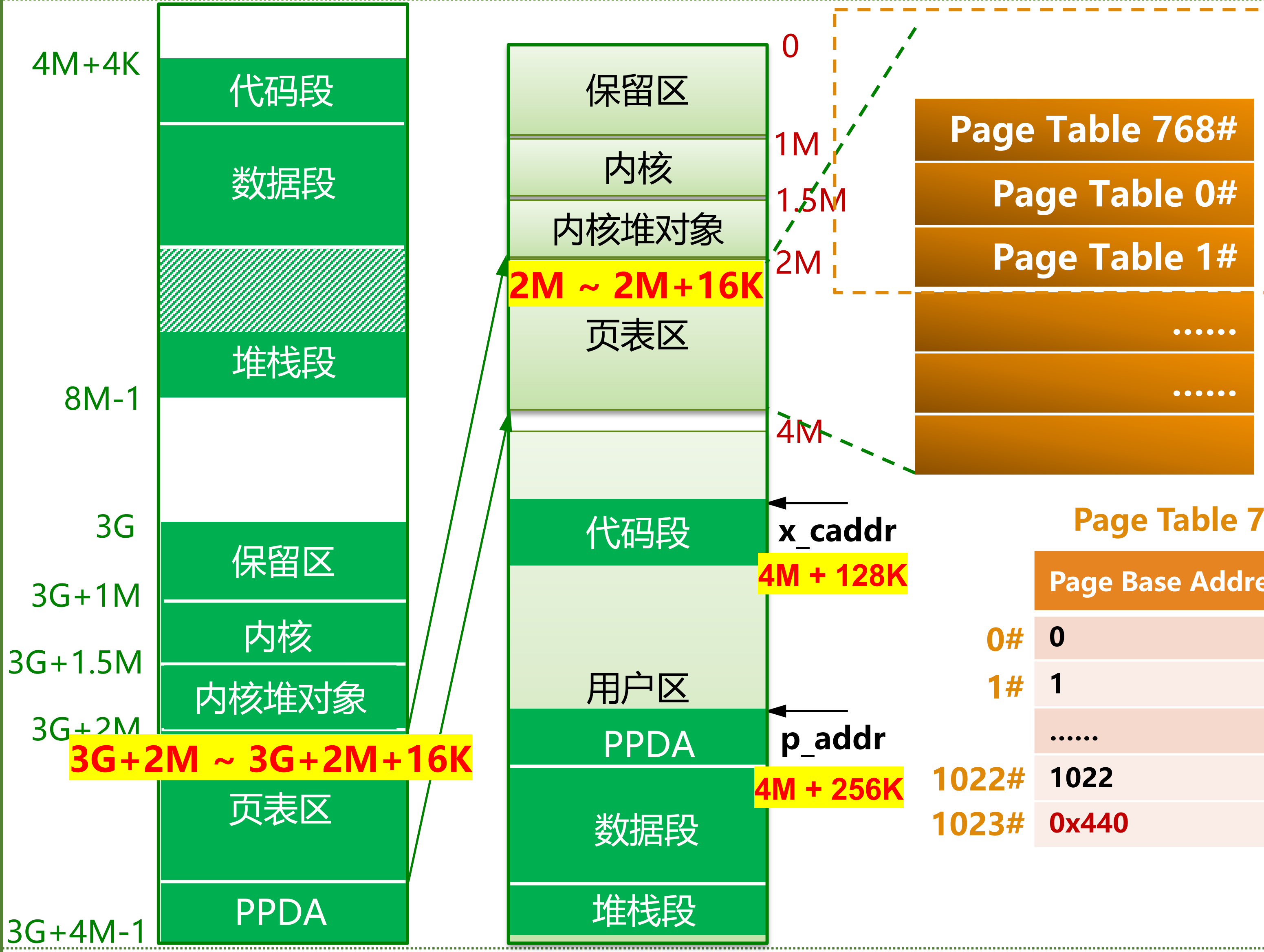


Page Table 768# （0x201号页框）

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

UNIX V6++的地址变换

现运行进程的页表



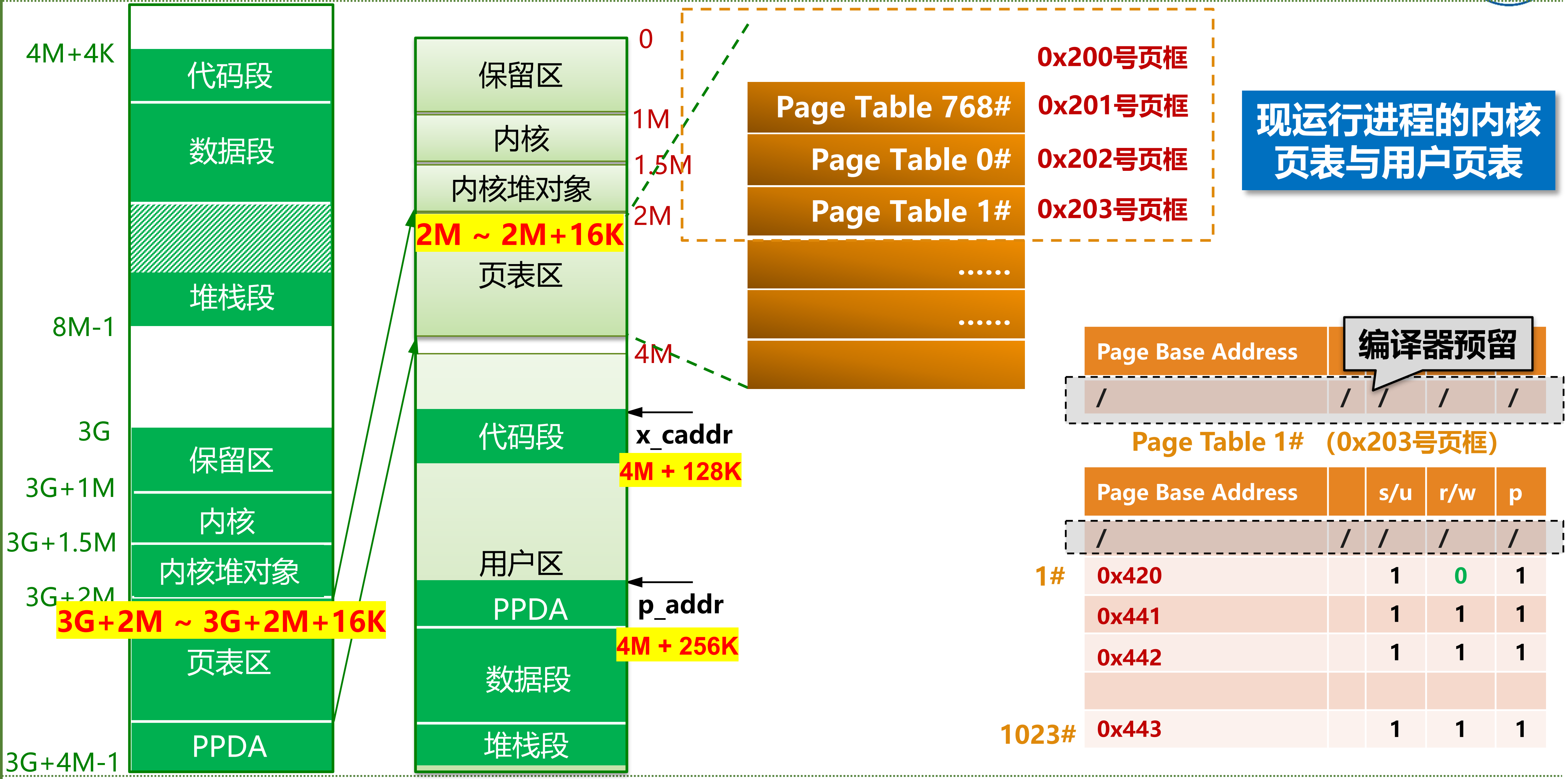
现运行进程的内核页表与用户页表

Page Table 768# （0x201号页框）

	Page Base Address		s/u	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1022#	1022		0	1	1
1023#	0x440		0	1	1

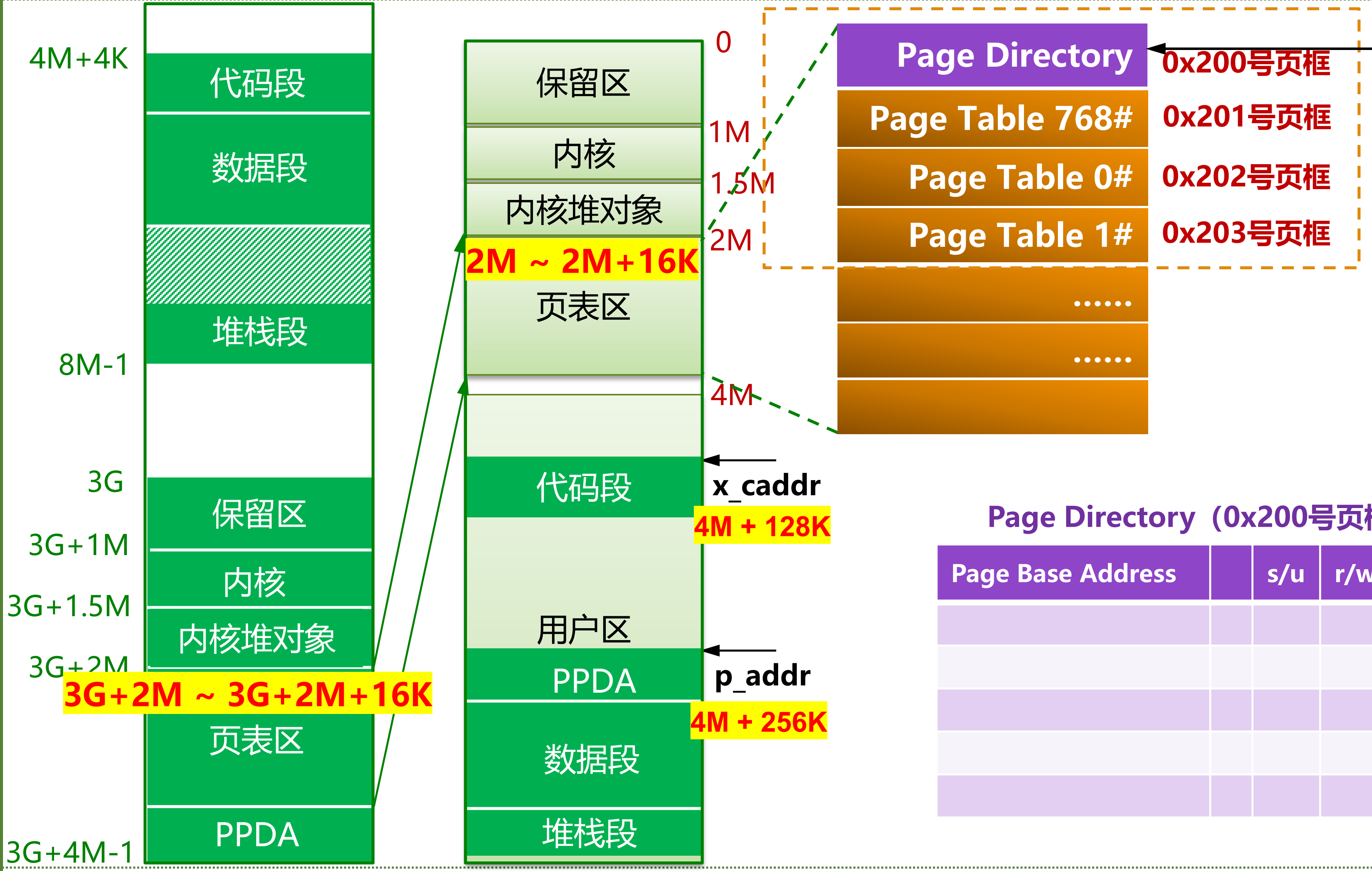
UNIX V6++的地址变换

现运行进程的页表



UNIX V6++的地址变换

现运行进程的页表



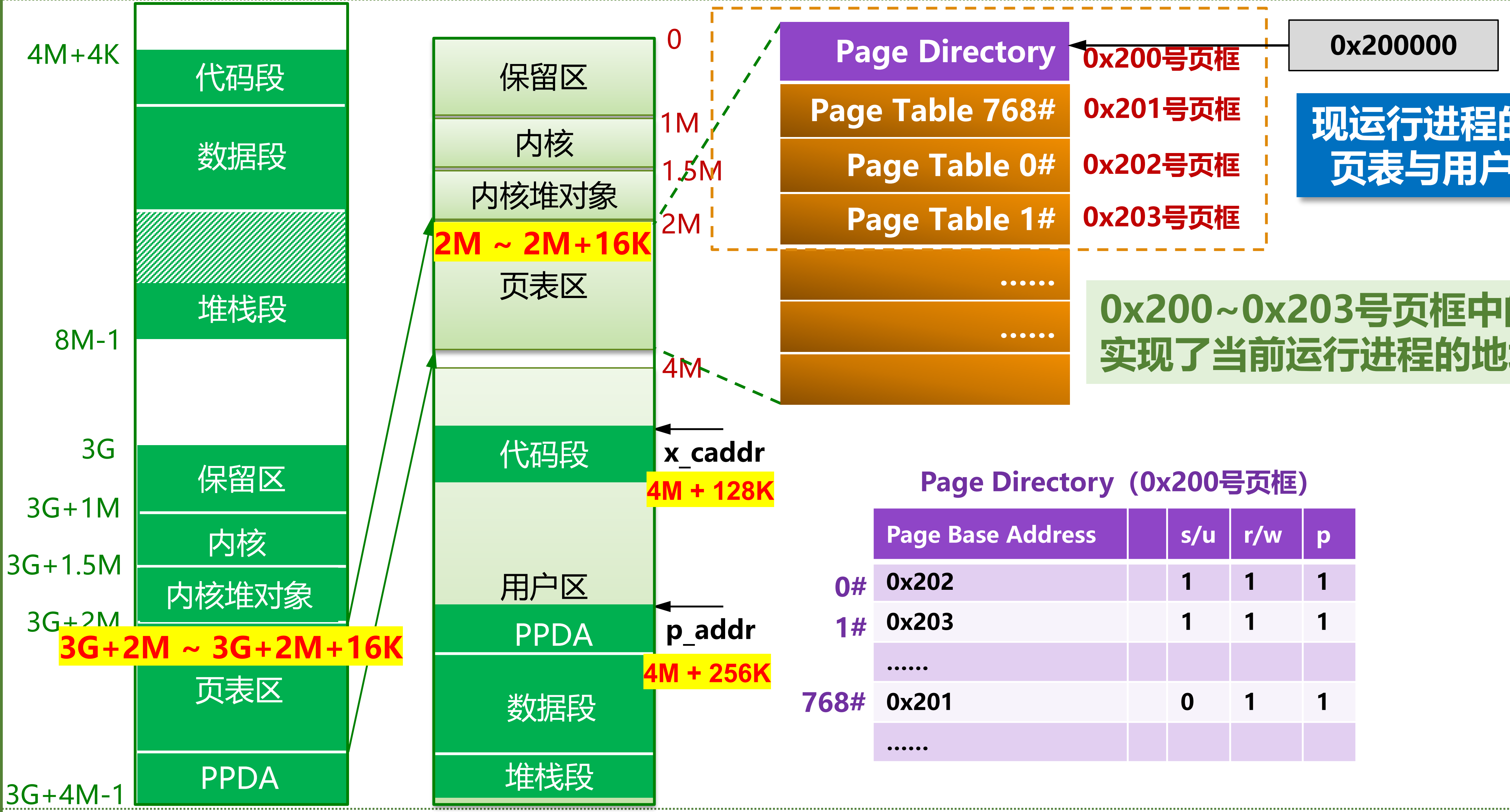
现运行进程的内核页表与用户页表

Page Directory（0x200号页框）

Page Base Address		s/u	r/w	p

UNIX V6++的地址变换

现运行进程的页表



现运行进程的内核页表与用户页表

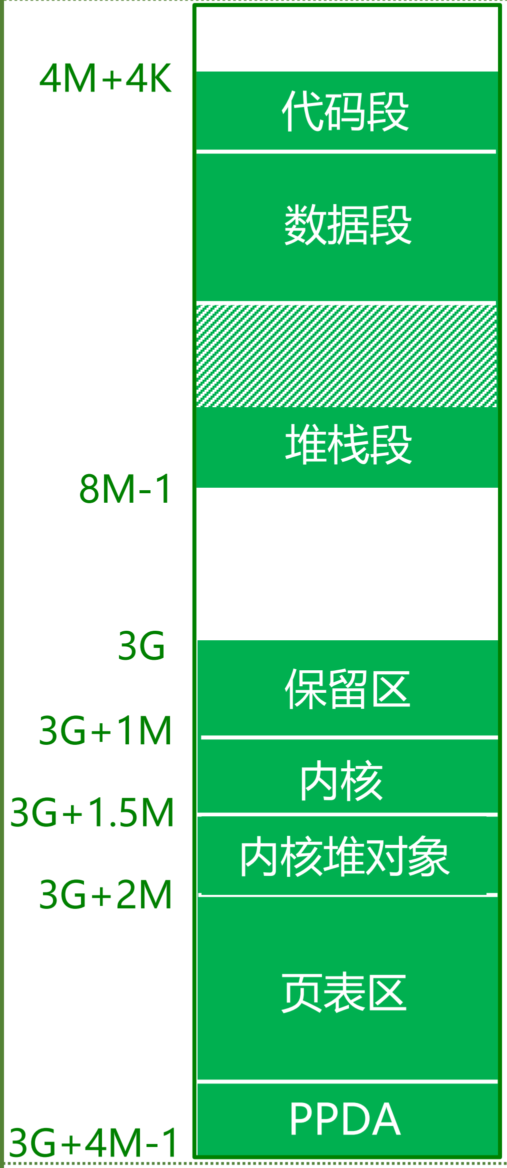
0x200~0x203号页框中的页表实现了当前运行进程的地址变换

Page Directory（0x200号页框）

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

UNIX V6++的地址变换

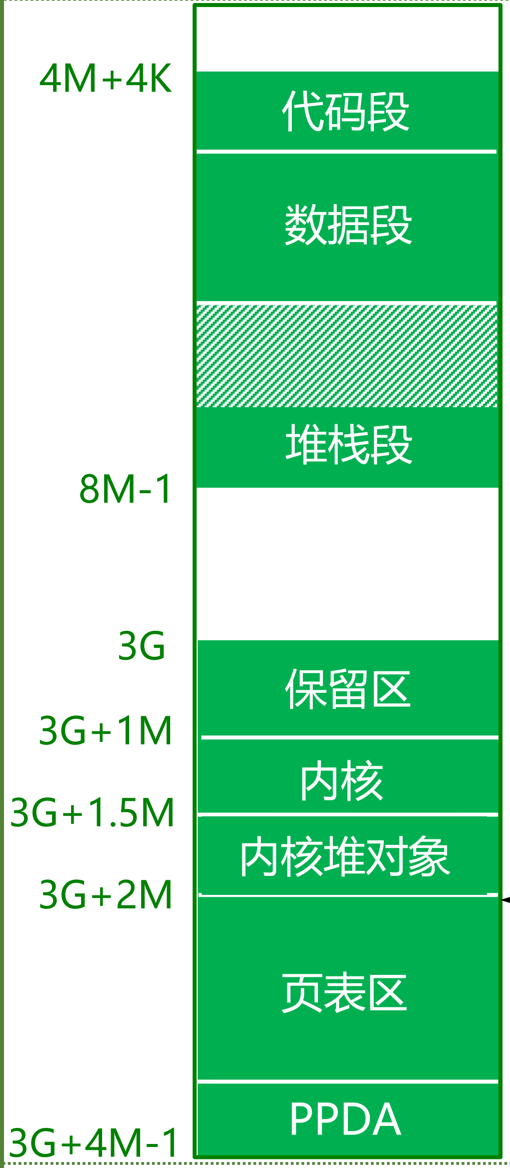
现运行进程的页表




```
class Machine
{
public:
    ……;
    /* 页目录、核心态页表、用户态页表在物理内存中的起始地址 */
    static const unsigned long PAGE_DIRECTORY_BASE_ADDRESS = 0x200000;
    static const unsigned long KERNEL_PAGE_TABLE_BASE_ADDRESS = 0x201000;
    static const unsigned long USER_PAGE_TABLE_BASE_ADDRESS = 0x202000;
    static const unsigned long USER_PAGE_TABLE_CNT = 2;
    /* 内核空间大小 4M 0xC0000000 - 0xC0400000 1 PageTable */
    static const unsigned int KERNEL_SPACE_SIZE = 0x400000;
    static const unsigned long KERNEL_SPACE_START_ADDRESS  = 0xC0000000;
public:
    void InitPageDirectory();
    void InitUserPageTable();
    PageDirectory& GetPageDirectory();          /* 获取当前正在使用的页目录表     */
    PageTable& GetKernelPageTable();            /* 获取操作系统内核所使用的页表，*/
    PageTable* GetUserPageTableArray();         /* 获取用户进程页表，共两张       */
    ……;
private:
    PageDirectory* m_PageDirectory;             /* 指向系统页目录 */
    PageTable*     m_KernelPageTable;           /* 指向内核页表   */
    PageTable*     m_UserPageTable;             /* 指向用户页表   */
    ……;
};
```

UNIX V6++的地址变换

页目录的初始化




```
void Machine::InitPageDirectory()
{
    PageDirectory* pPageDirectory = (PageDirectory*)
                        (PAGE_DIRECTORY_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
    ……;
```

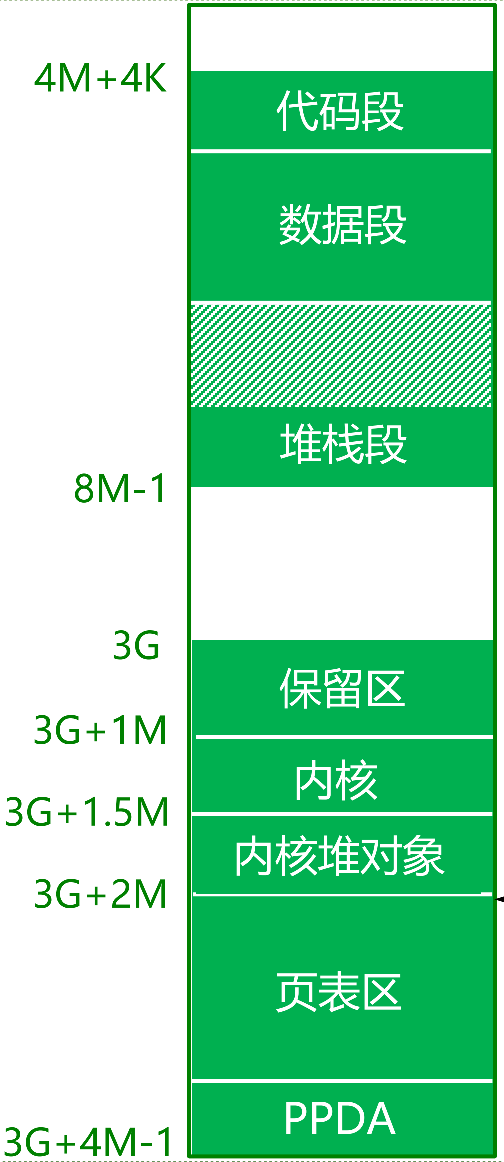
PAGE_DIRECTORY_BASE_ADDRESS: 0x200000

KERNEL_SPACE_START_ADDRESS: 3G

pPageDirectory

UNIX V6++的地址变换

页目录的初始化




```
void Machine::InitPageDirectory()
{
    PageDirectory* pPageDirectory = (PageDirectory*)
                                  (PAGE_DIRECTORY_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
    ……;
    /* 填写200#页表的第768项 */
    unsigned int kPageTableIdx = KERNEL_SPACE_START_ADDRESS /
                                  PageTable::SIZE_PER_PAGETABLE_MAP;
    pPageDirectory->m_Entrys[kPageTableIdx].m_UserSupervisor = 0;
    pPageDirectory->m_Entrys[kPageTableIdx].m_Present = 1;
    pPageDirectory->m_Entrys[kPageTableIdx].m_ReadWriter = 1;
    pPageDirectory->m_Entrys[kPageTableIdx].m_PageTableBaseAddress =
                                  KERNEL_PAGE_TABLE_BASE_ADDRESS >> 12;
```

3G/4M

0x201

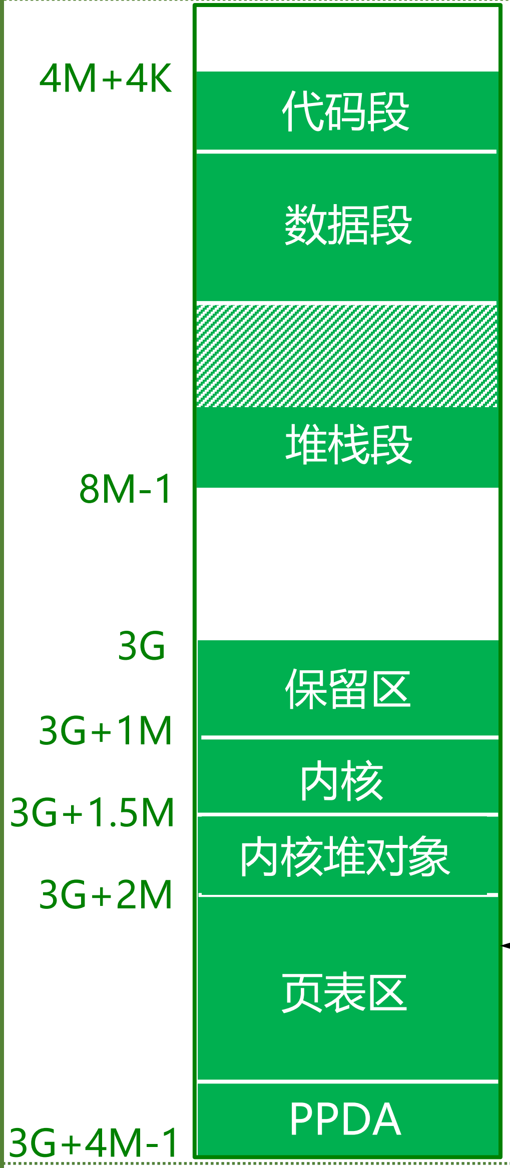
pPageDirectory

Page Directory（0x200号页框）

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

UNIX V6++的地址变换

页目录的初始化



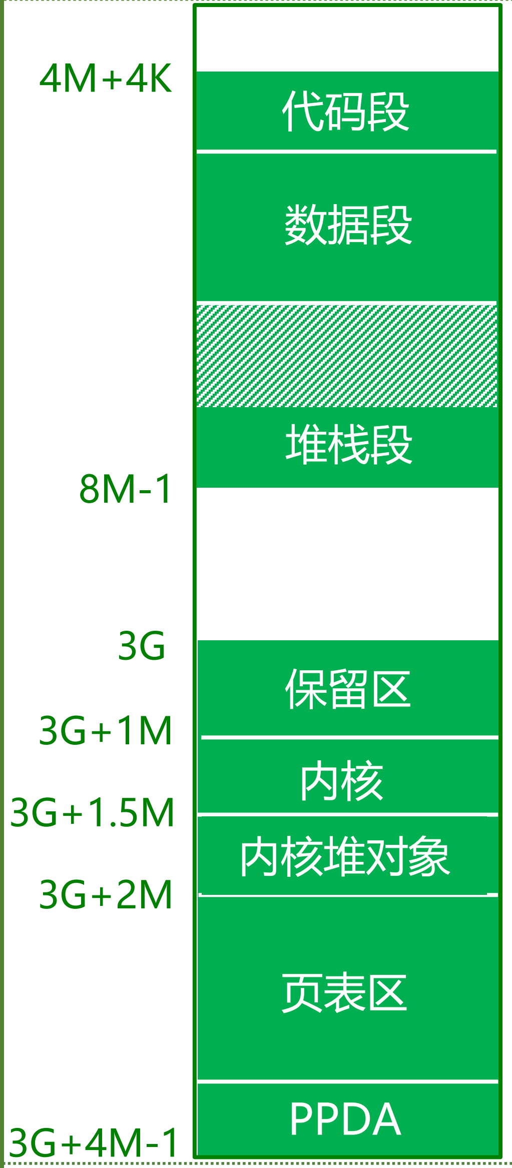

```
void Machine::InitPageDirectory()
{
    PageDirectory* pPageDirectory = (PageDirectory*)
                              (PAGE_DIRECTORY_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
    ……;
    /* 填写200#页表的第768项 */
    unsigned int kPageTableIdx = KERNEL_SPACE_START_ADDRESS /
                              PageTable::SIZE_PER_PAGETABLE_MAP;
    pPageDirectory->m_Entrys[kPageTableIdx].m_UserSupervisor = 0;
    pPageDirectory->m_Entrys[kPageTableIdx].m_Present = 1;
    pPageDirectory->m_Entrys[kPageTableIdx].m_ReadWriter = 1;
    pPageDirectory->m_Entrys[kPageTableIdx].m_PageTableBaseAddress =
                              KERNEL_PAGE_TABLE_BASE_ADDRESS >> 12;
    PageTable* pPageTable = (PageTable*)
                    (KERNEL_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
```

$\underbrace{\text{KERNEL\_PAGE\_TABLE\_BASE\_ADDRESS}}_{\text{0x201000}}$ + $\underbrace{\text{KERNEL\_SPACE\_START\_ADDRESS}}_{\text{3G}}$

pPageTable

UNIX V6++的地址变换

页目录的初始化



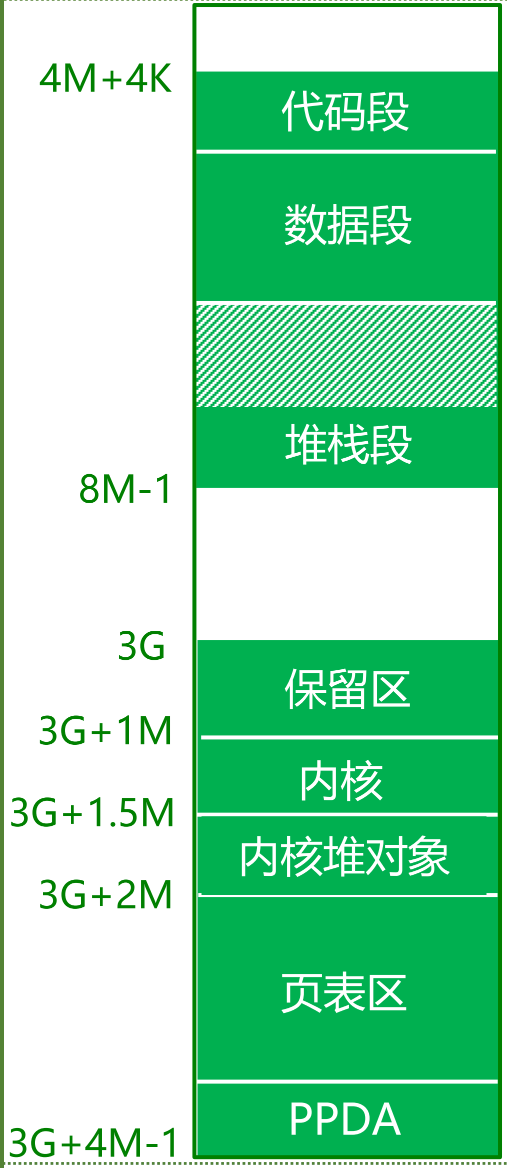

```
void Machine::InitPageDirectory()
{
    PageDirectory* pPageDirectory = (PageDirectory*)
                                (PAGE_DIRECTORY_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
    ......;
    /* 填写200#页表的第768项 */
    unsigned int kPageTableIdx = KERNEL_SPACE_START_ADDRESS /
                                        PageTable::SIZE_PER_PAGETABLE_MAP;
    pPageDirectory->m_Entrys[kPageTableIdx].m_UserSupervisor = 0;
    pPageDirectory->m_Entrys[kPageTableIdx].m_Present = 1;
    pPageDirectory->m_Entrys[kPageTableIdx].m_ReadWriter = 1;
    pPageDirectory->m_Entrys[kPageTableIdx].m_PageTableBaseAddress =
                                        KERNEL_PAGE_TABLE_BASE_ADDRESS >> 12;
    PageTable* pPageTable = (PageTable*)
                        (KERNEL_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
    for ( unsigned int i = 0; i < PageTable::ENTRY_CNT_PER_PAGETABLE; i++ )
    {
        pPageTable->m_Entrys[i].m_UserSupervisor = 0;
        pPageTable->m_Entrys[i].m_Present = 1;
        pPageTable->m_Entrys[i].m_ReadWriter = 1;
        pPageTable->m_Entrys[i].m_PageBaseAddress = i;
    }
        this->m_PageDirectory = pPageDirectory;
        this->m_KernelPageTable = pPageTable;
}
```

Page Table 768# （0x201号页框）

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

UNIX V6++的地址变换

用户页表的初始化



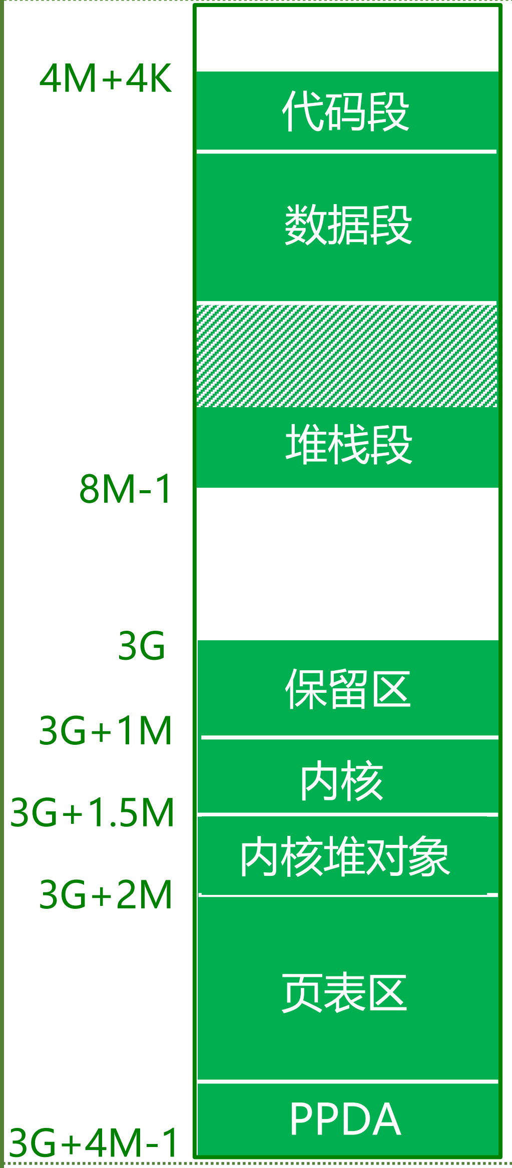

```
void Machine::InitUserPageTable()
{
    PageDirectory* pPageDirectory = this->m_PageDirectory;
    PageTable* pUserPageTable = (PageTable*)
                        (USER_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);
```

USER_PAGE_TABLE_BASE_ADDRESS: 0x202000

KERNEL_SPACE_START_ADDRESS: 3G

UNIX V6++的地址变换

用户页表的初始化



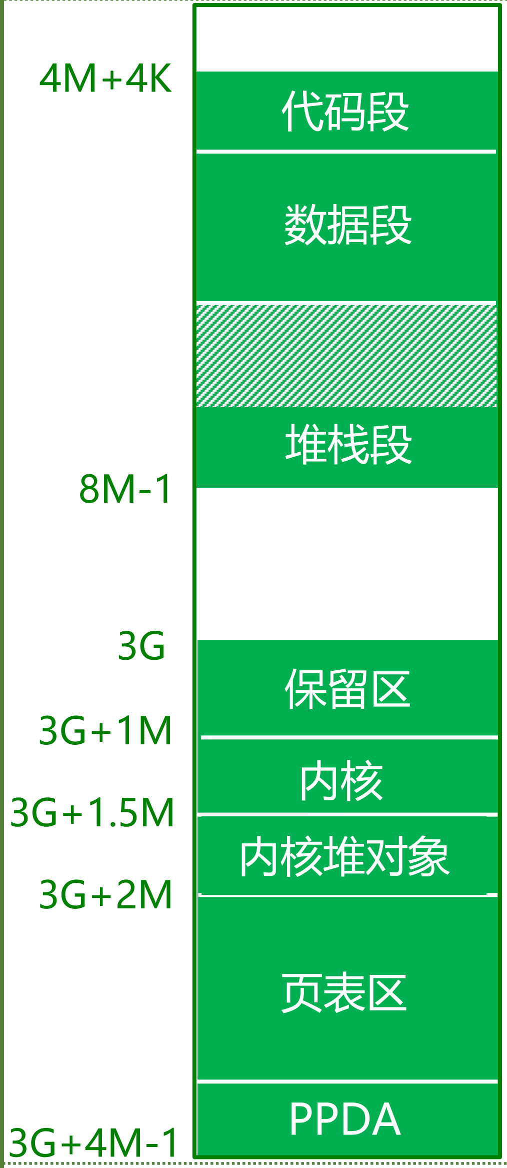

```
void Machine::InitUserPageTable()
{
    PageDirectory* pPageDirectory = this->m_PageDirectory;
    PageTable* pUserPageTable = (PageTable*)
                                (USER_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);

    unsigned int idx = USER_PAGE_TABLE_BASE_ADDRESS >> 12;   0x202
    for ( unsigned int j = 0; j < USER_PAGE_TABLE_CNT; j++, idx++ )
    {
        pPageDirectory->m_Entrys[j].m_UserSupervisor = 1;
        pPageDirectory->m_Entrys[j].m_Present = 1;
        pPageDirectory->m_Entrys[j].m_ReadWriter = 1;
        pPageDirectory->m_Entrys[j].m_PageTableBaseAddress = idx;
    }
}
```

Page Directory（0x200号页框）

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

UNIX V6++的地址变换

用户页表的初始化



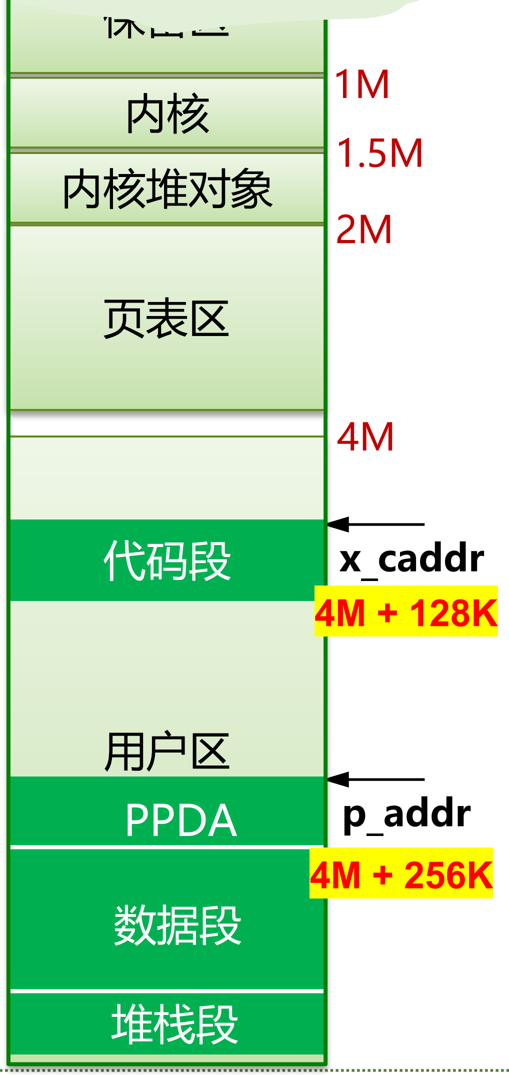

```
void Machine::InitUserPageTable()
{
    PageDirectory* pPageDirectory = this->m_PageDirectory;
    PageTable* pUserPageTable = (PageTable*)
                              (USER_PAGE_TABLE_BASE_ADDRESS + KERNEL_SPACE_START_ADDRESS);

    unsigned int idx = USER_PAGE_TABLE_BASE_ADDRESS >> 12;
    for ( unsigned int j = 0; j < USER_PAGE_TABLE_CNT; j++, idx++ )
    {
        pPageDirectory->m_Entrys[j].m_UserSupervisor = 1;
        pPageDirectory->m_Entrys[j].m_Present = 1;
        pPageDirectory->m_Entrys[j].m_ReadWriter = 1;
        pPageDirectory->m_Entrys[j].m_PageTableBaseAddress = idx;

        for ( unsigned int i = 0; i < PageTable::ENTRY_CNT_PER_PAGETABLE; i++ )
        {
            pUserPageTable[j].m_Entrys[i].m_UserSupervisor = 1;
            pUserPageTable[j].m_Entrys[i].m_Present = 1;
            pUserPageTable[j].m_Entrys[i].m_ReadWriter = 1;
            pUserPageTable[j].m_Entrys[i].m_PageBaseAddress = 0x00000 + i + j * 1024;
        }
    }
    this->m_UserPageTable = pUserPageTable;
}
```

1024

当前内核初始化阶段并不使用这段内存，也不会向4M-8M物理内存写入任何数据。

UNIX V6++的地址变换

进程取指时：



EIP内的地址为：4M+7k

CR3: 0x200000

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		s/u	r/w	p
0x202		1	1	1
0x203		1	1	1
……				
0x201		0	1	1
……				

Page Table 1#（0x203号页框）

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

UNIX V6++的地址变换

执行指令时：

内核 1M
内核堆对象 1.5M
2M
页表区
4M
代码段 ← x_caddr
M + 128K
用户区
PPDA ← p_addr
M + 256K
数据段
堆栈段

0x200000 CR3

取回的指令为：inc [4M+14k]

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		s/u	r/w	p
0x202		1	1	1
0x203		1	1	1
……				
0x201		0	1	1
……				

1

3

Page Table 1#　（0x203号页框）

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

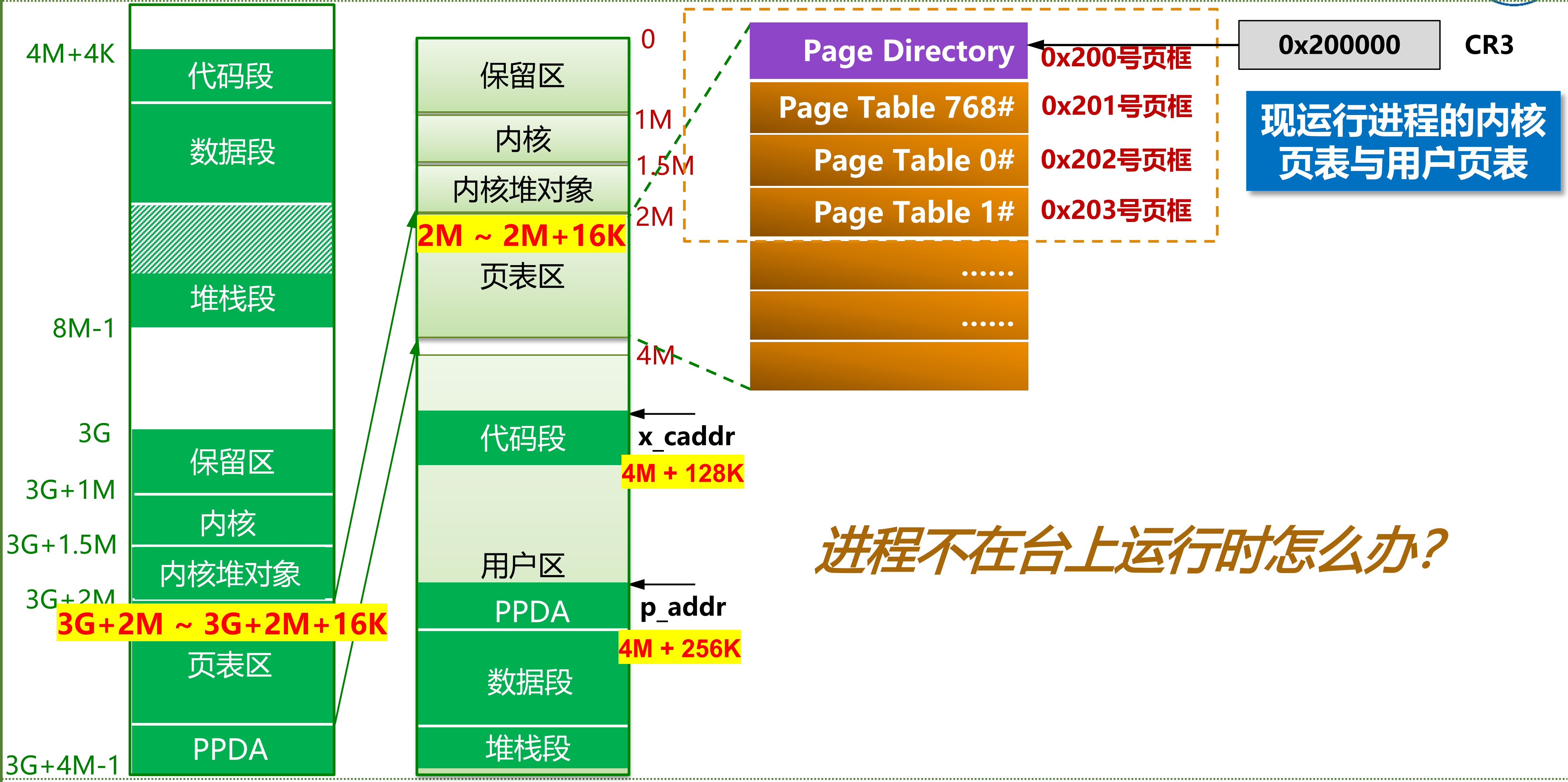
4M+266K

00 00 0000 01 00 0100 0010	10 00 0000 0000

另一种计算方法： 4M+8K ⟶ 4M+260K　4M+14K ⟶ 4M+266K

UNIX V6++的地址变换

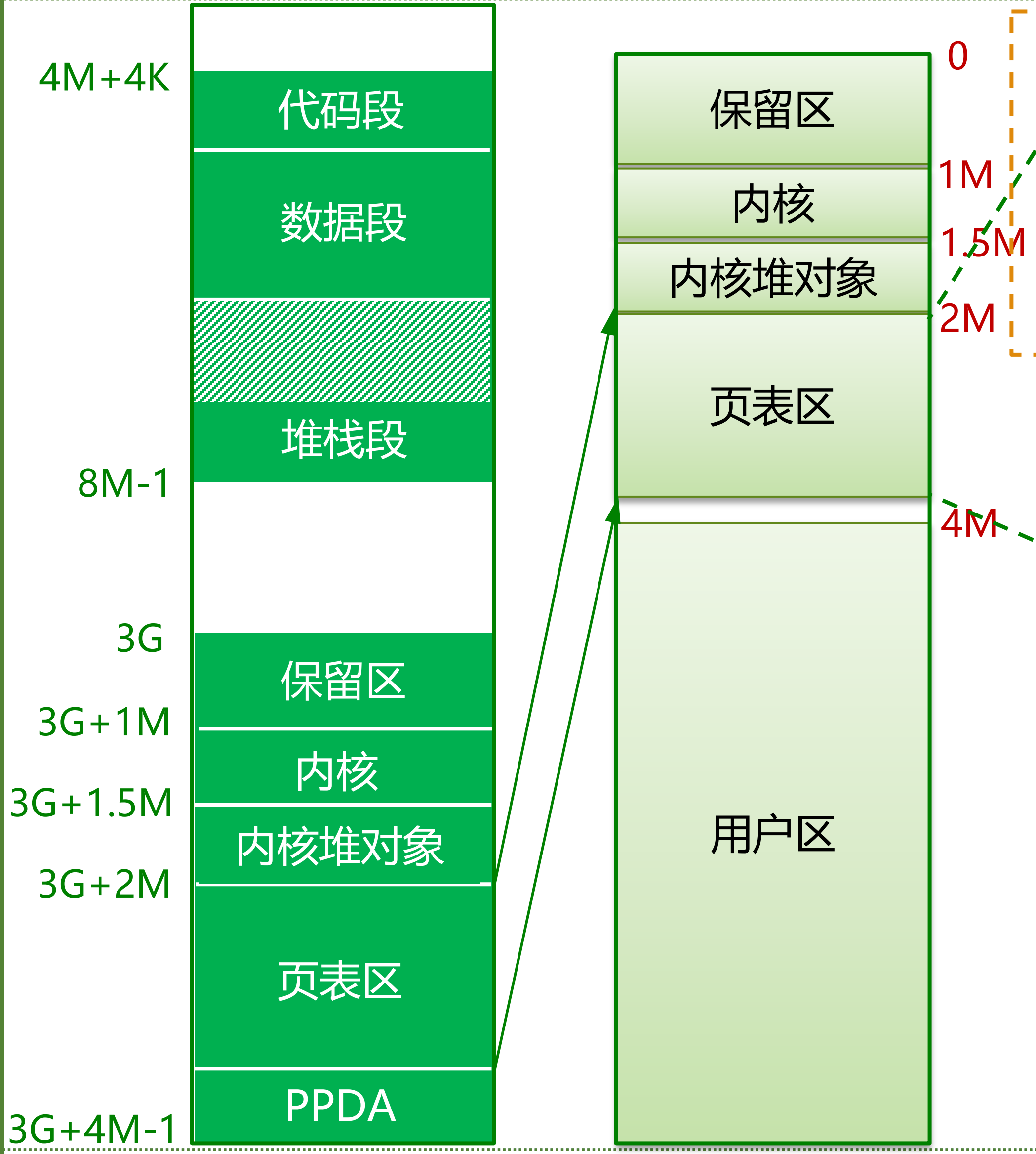
其他进程的页表



进程不在台上运行时怎么办?

UNIX V6++的地址变换

其他进程的页表



现运行进程的内核页表与用户页表

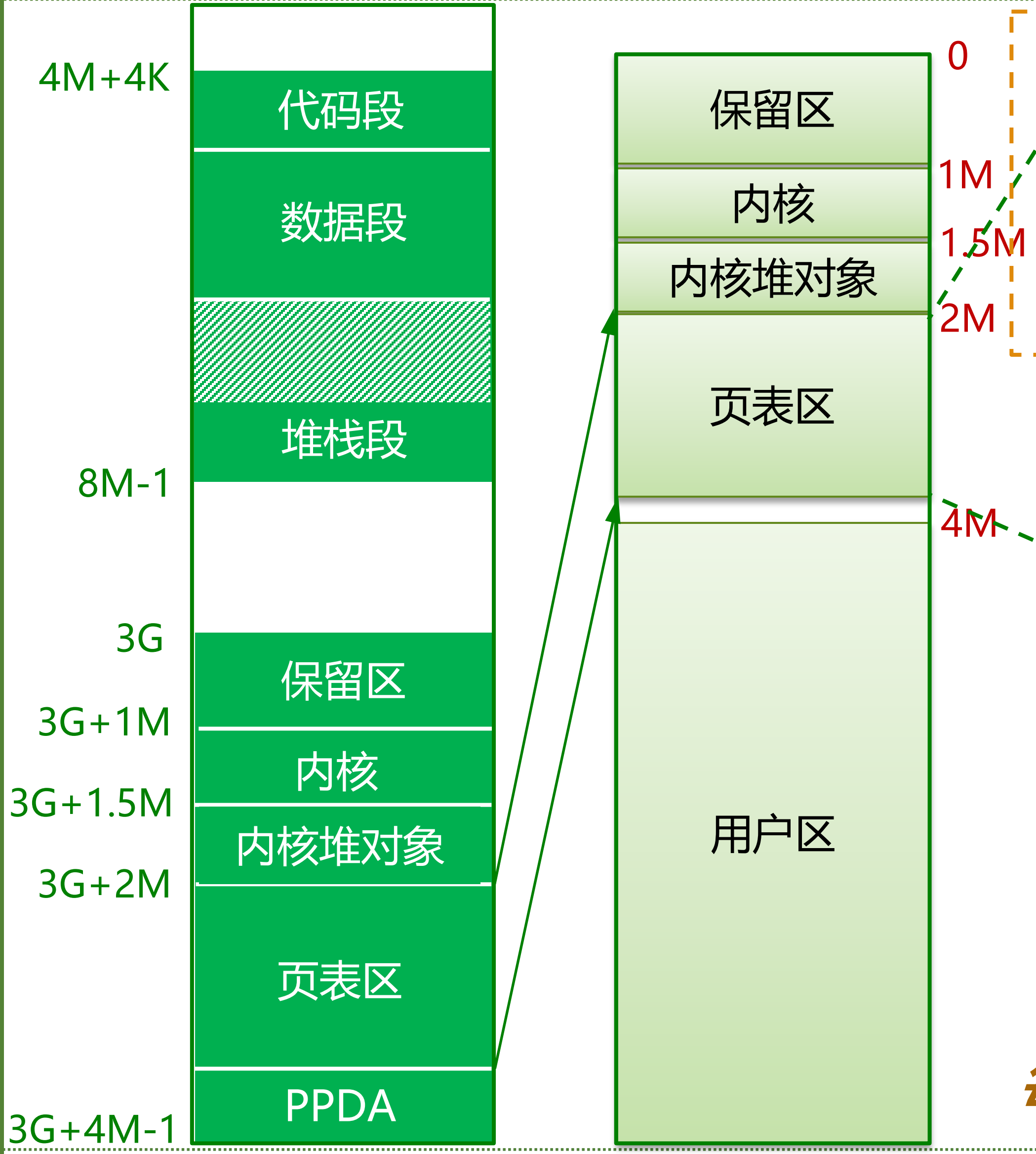
对所有进程都是一样的
（*进程切换调度时不需修改*）

Page Directory（0x200号页框）

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

UNIX V6++的地址变换

其他进程的页表



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*）

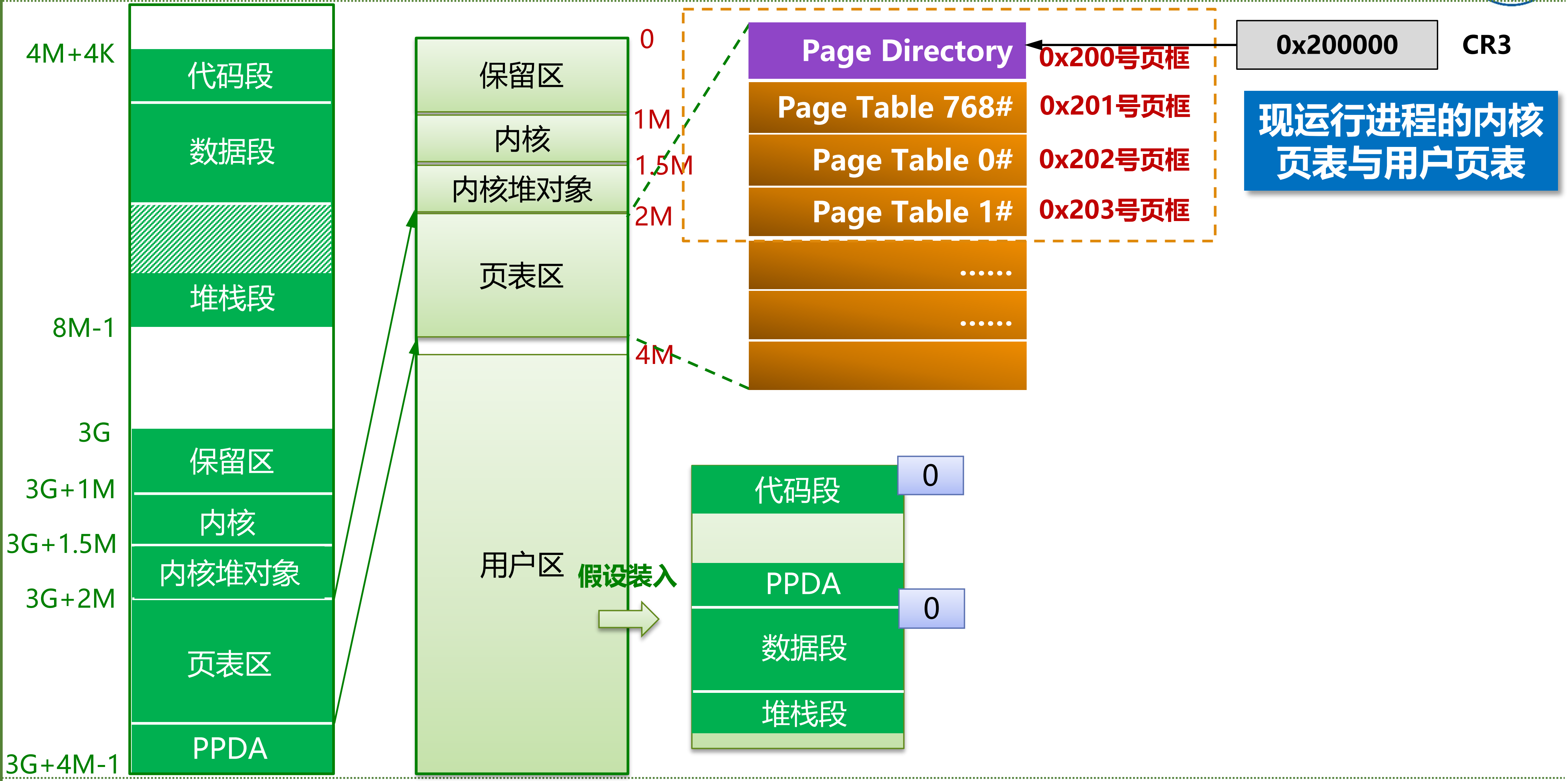
Page Table 768#　(0x201号页框)

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

每个页目录和核心页表进程共用，节省空间

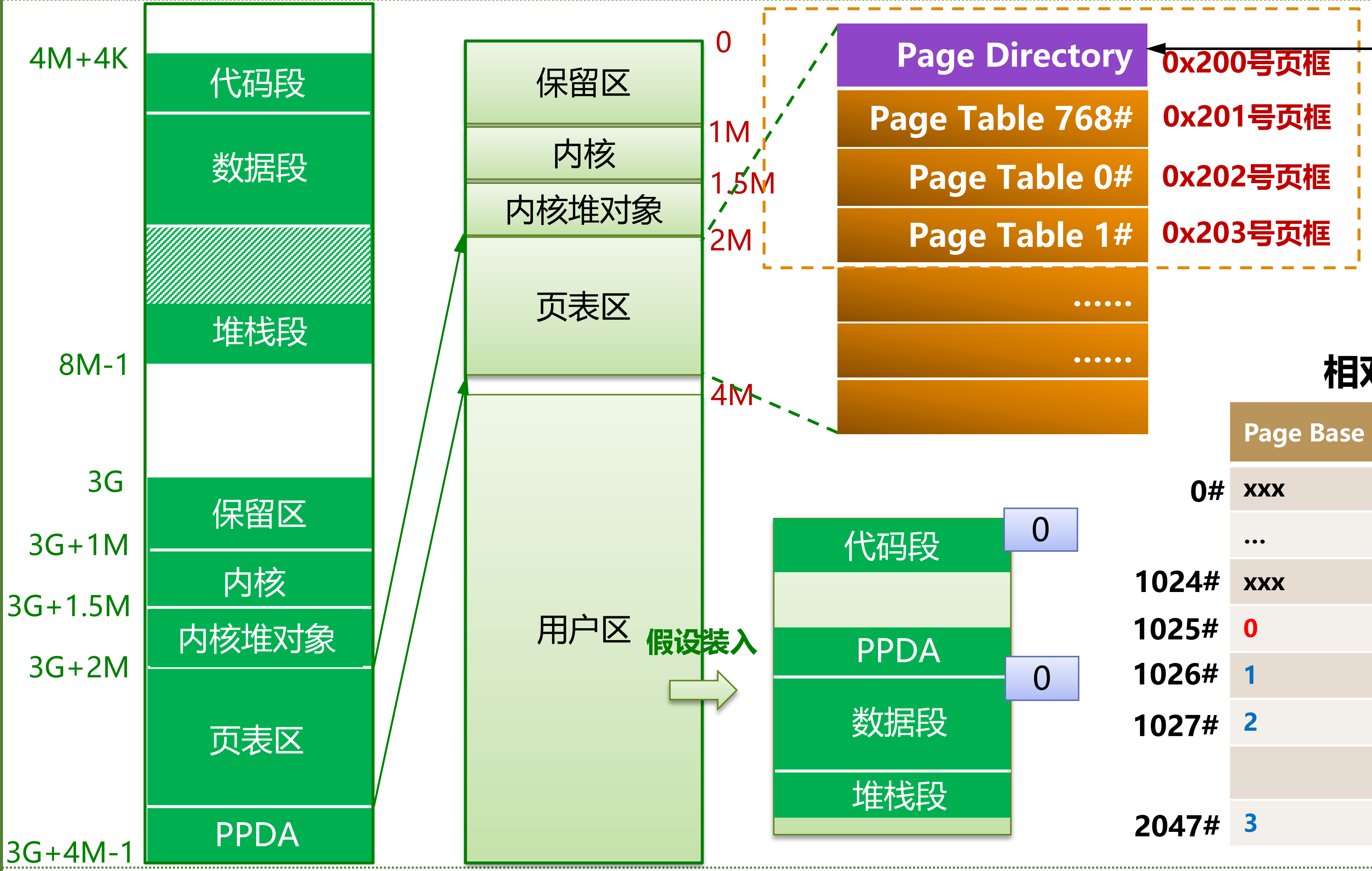
UNIX V6++的地址变换

其他进程的页表



UNIX V6++的地址变换

其他进程的页表



0xC0200000 CR3

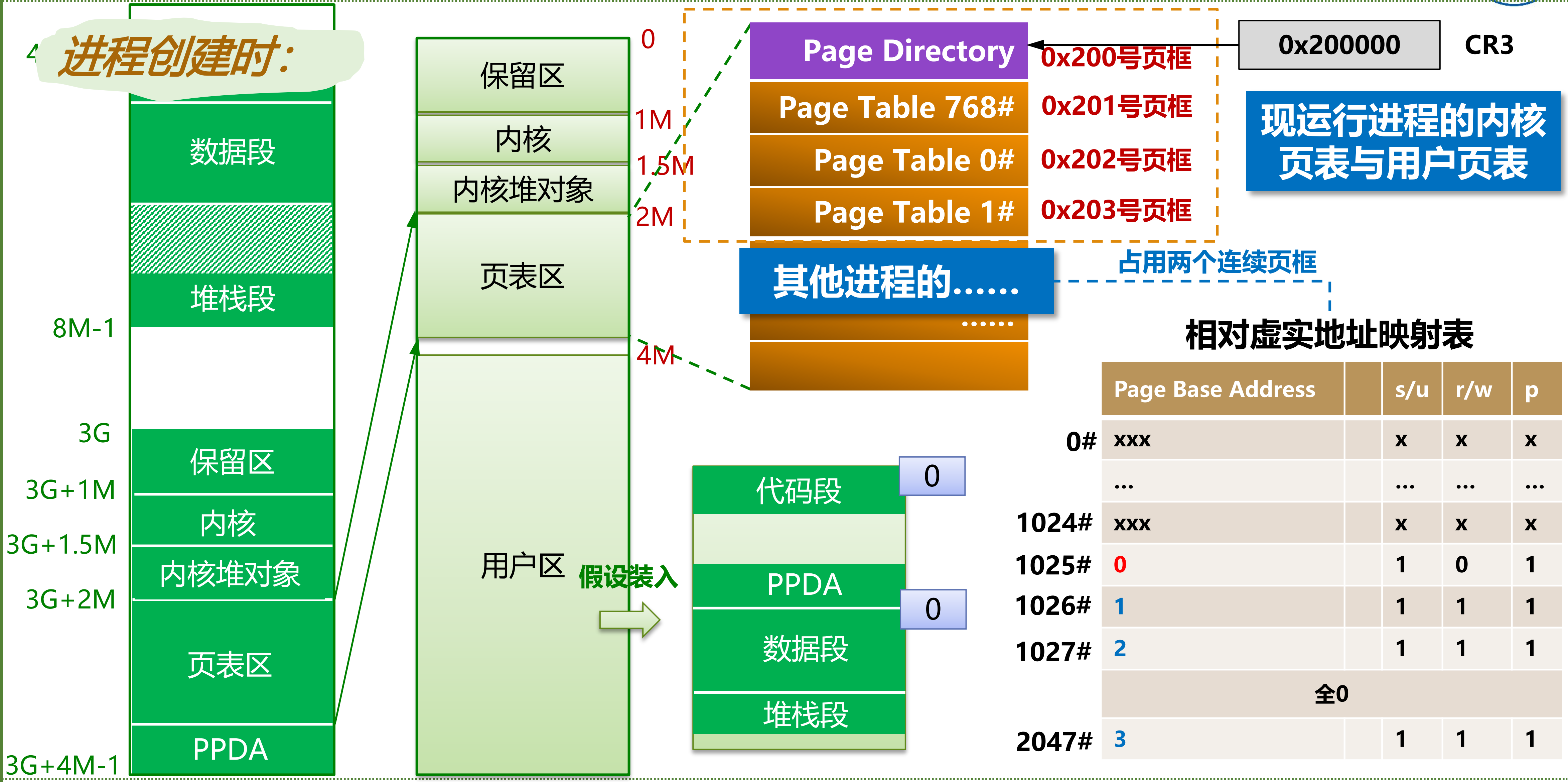
现运行进程的内核页表与用户页表

相对虚实地址映射表

	Page Base Address		s/u	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	1	1
1027#	2		1	1	1
	全0				
2047#	3		1	1	1

UNIX V6++的地址变换

其他进程的页表



相对虚实地址映射表

	Page Base Address		s/u	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	1	1
1027#	2		1	1	1
	全0				
2047#	3		1	1	1

UNIX V6++的地址变换

相对虚实地址映射表

进程创建时：

相对虚实地址映射表

	Page Base Address		s/u	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	1	1
1027#	2		1	1	1
	全0				
2047#	3		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

```
class MemoryDescriptor
{
public:
        void Initialize();     /* 进程创建时申请空白相对虚实地址映射表*/
                               /* 逻辑址记入m_UserPageTableArray */
        void Release();        /* 进程终止时，释放相对虚实地址映射表 */
        void ClearUserPageTable();   /* 清理相对虚实地址映射表 */

        /* 根据各部分的起始逻辑地址和大小构建相对虚实地址映射表 */
        bool EstablishUserPageTable(unsigned long textVirtualAddress,
                                    unsigned long textSize,
                                    unsigned long dataVirtualAddress,
                                    unsigned long dataSize,
                                    unsigned long stackSize);
        /* 根据相对虚实地址映射表构建物理页表 */
        void MapToPageTable();
        ……;
public:
        PageTable*        m_UserPageTableArray;
        unsigned long     m_TextStartAddress;    /* 代码段起始地址 */
        unsigned long     m_TextSize;            /* 代码段长度 */
        unsigned long     m_DataStartAddress;    /* 数据段起始地址 */
        unsigned long     m_DataSize;            /* 数据段长度 */
        unsigned long     m_StackSize;           /* 栈段长度 */
};
```

UNIX V6++的地址变换

进程创建时：

相对虚实地址映射表

	Page Base Address		s/u	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1025#	0		1	0	1
1026#	1		1	1	1
1027#	2		1	1	1
	全0				
2047#	3		1	1	1

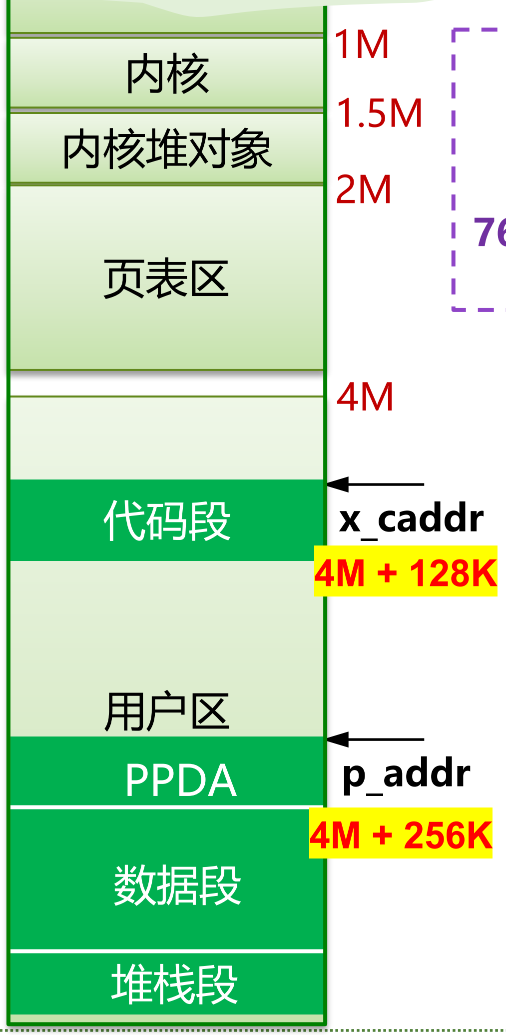
每次换进/换出，只需修改p_addr和x_caddr，无需刷新页表，节省时间

```
class MemoryDescriptor
{
public:
    void Initialize();    /* 进程创建时申请空白相对虚实地址映射表*/
                          /* 逻辑址记入m_UserPageTableArray */
    void Release();       /* 进程终止时，释放相对虚实地址映射表 */
    void ClearUserPageTable();   /* 清理相对虚实地址映射表 */

    /* 根据各部分的起始逻辑地址和大小构建相对虚实地址映射表 */
    bool EstablishUserPageTable(unsigned long textVirtualAddress,
                                unsigned long textSize,
                                unsigned long dataVirtualAddress,
                                unsigned long dataSize,
                                unsigned long stackSize);
    /* 根据相对虚实地址映射表构建物理页表 */
    void MapToPageTable();
    ……;
public:
    PageTable*       m_UserPageTableArray;
    unsigned long    m_TextStartAddress; =4M+4K段起始地址 */
    unsigned long    m_TextSize; =4K        /* 代码段长度 */
    unsigned long    m_DataStartAddress; =4M+8K段起始地址 */
    unsigned long    m_DataSize; =8K        /* 数据段长度 */
    unsigned long    m_StackSize; =4K       /* 栈段长度 */
};
```

UNIX V6++的地址变换

相对虚实地址映射表

进程pa上台时:



Page Directory（0x200号页框）

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

Page Table 768# （0x201号页框）

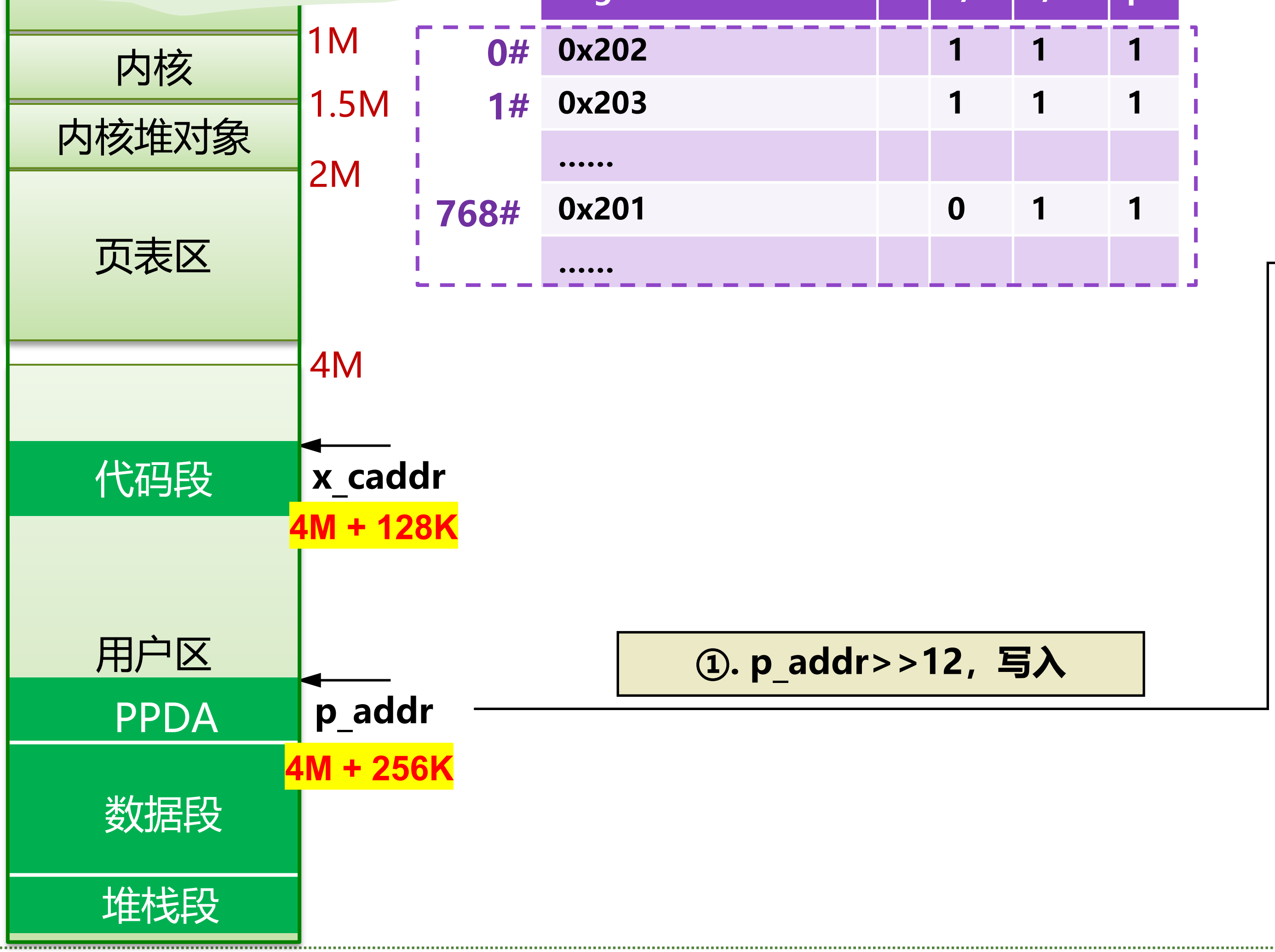
	Page Base Address		s/u	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1022#	1022		0	1	1
1023#			0	1	1

对所有进程都是一样的
（*进程切换调度时不需修改*）

UNIX V6++的地址变换

相对虚实地址映射表

进程上台时：



Page Directory（0x200号页框）

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

Page Table 768# （0x201号页框）

	Page Base Address		s/u	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1022#	1022		0	1	1
1023#	0x440		0	1	1

UNIX V6++的地址变换

相对虚实地址映射表

Page Directory（0x200号页框）

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

Page Table 768# （0x201号页框）

	Page Base Address		s/u	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1022#	1022		0	1	1
1023#	0x440		0	1	1

Page Table 0# （0x202号页框）

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

Page Table 1# （0x203号页框）

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

相对虚实地址映射表

	Page Base Address		s/u	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		1	1	1
	全0				
2047#	3		1	1	1

②. if r/w==0, x_caddr>>12，加上PBA

相对虚实地址映射表

UNIX V6++的地址变换

相对虚实地址映射表

Page Directory（0x200号页框）

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

Page Table 768#（0x201号页框）

	Page Base Address		s/u	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1022#	1022		0	1	1
1023#	0x440		0	1	1

相对虚实地址映射表

	Page Base Address		s/u	r/w	p
0#	xxx		x	x	x
	…		…	…	…
1024#	xxx		x	x	x
1[illegible]					
1026#	1		1	0	1
1027#	2		1	1	1
	全0				
2047#	3		1	1	1

③. if r/w==1, p_addr>>12，加上PBA

Page Table 0#（0x202号页框）

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

Page Table 1#（0x203号页框）

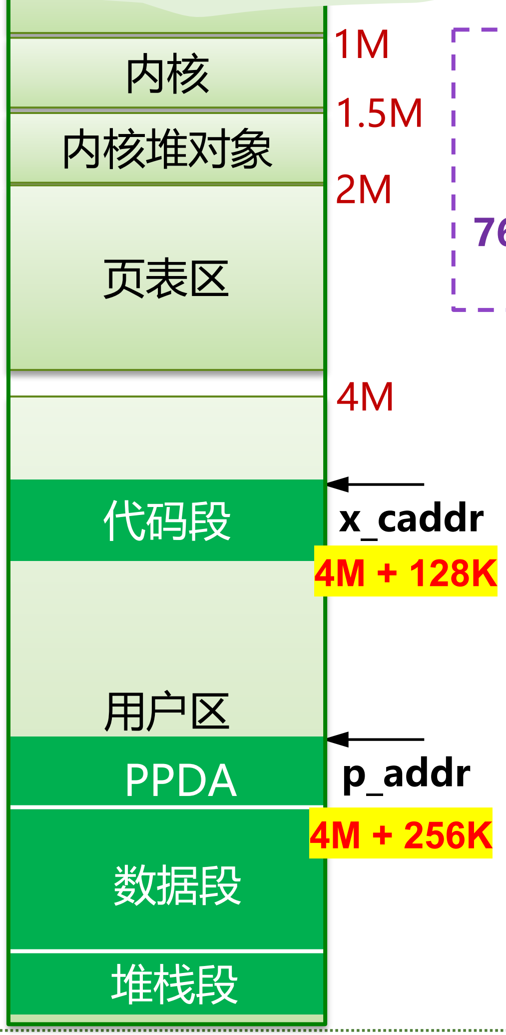
	Page Base Address		s/u	r/w	p
0#	/	/	/	/	/
1#	0x420		1	0	1
	0x441		1	1	1
	0x442		1	1	1
1023#	0x443		1	1	1

相对虚实地址映射表

UNIX V6++的地址变换

相对虚实地址映射表

进程上台时:



Page Directory（0x200号页框）

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

Page Table 768# （0x201号页框）

	Page Base Address		s/u	r/w	p
0#	0		0	1	1
1#	1		0	1	1
	……				
1022#	1022		0	1	1
1023#	0x440		0	1	1

Page Table 0# （0x202号页框）

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

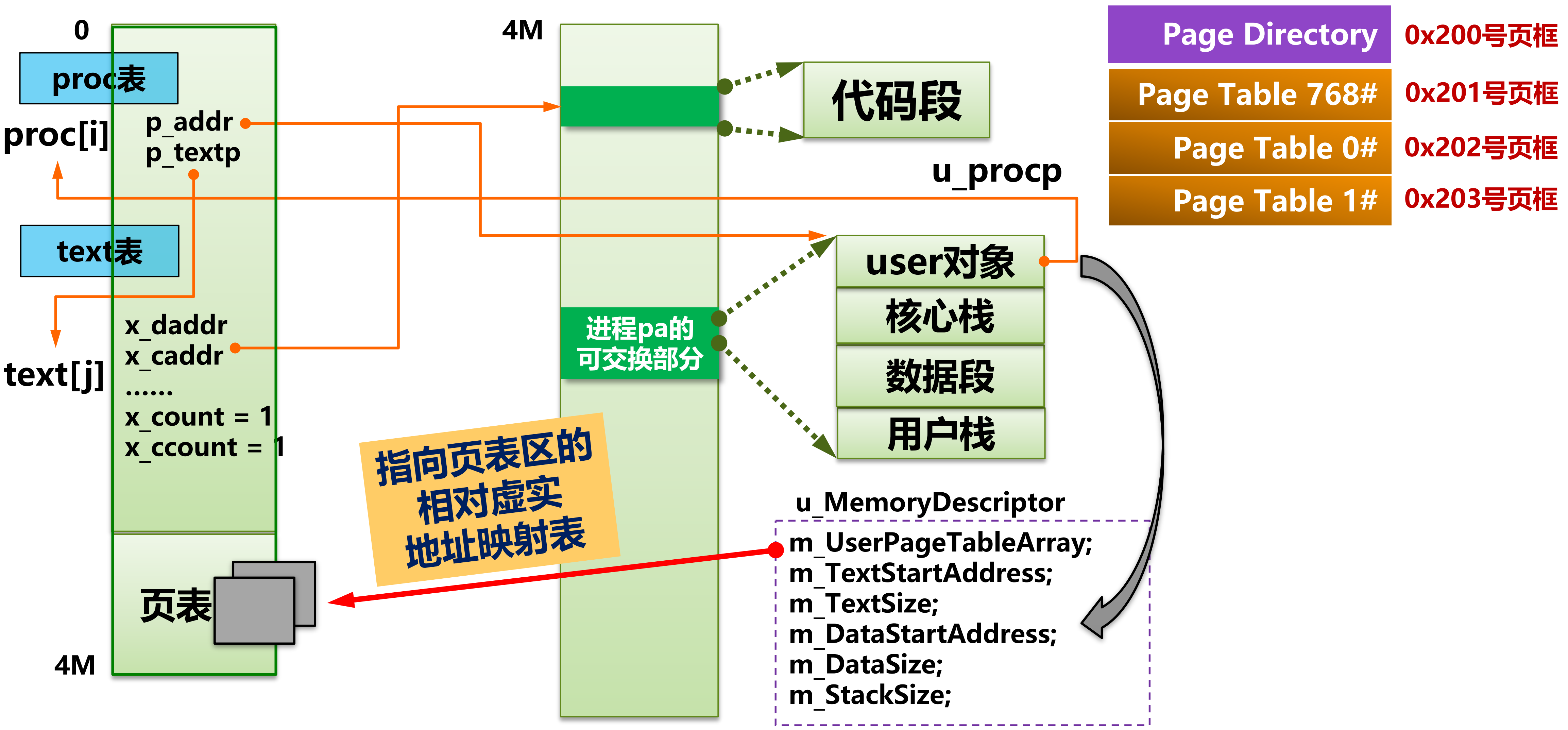
Page Table 1# （0x203号页框）

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

相对虚实地址映射表

	Page Base Address		s/u	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		1	1	1
	全0				
2047#	3		1	1	1

现运行进程完整的进程图像



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

阅读教材：154页 ~ 164页