

# 进程管理 设计与实现

ZJUNIX



## 目录

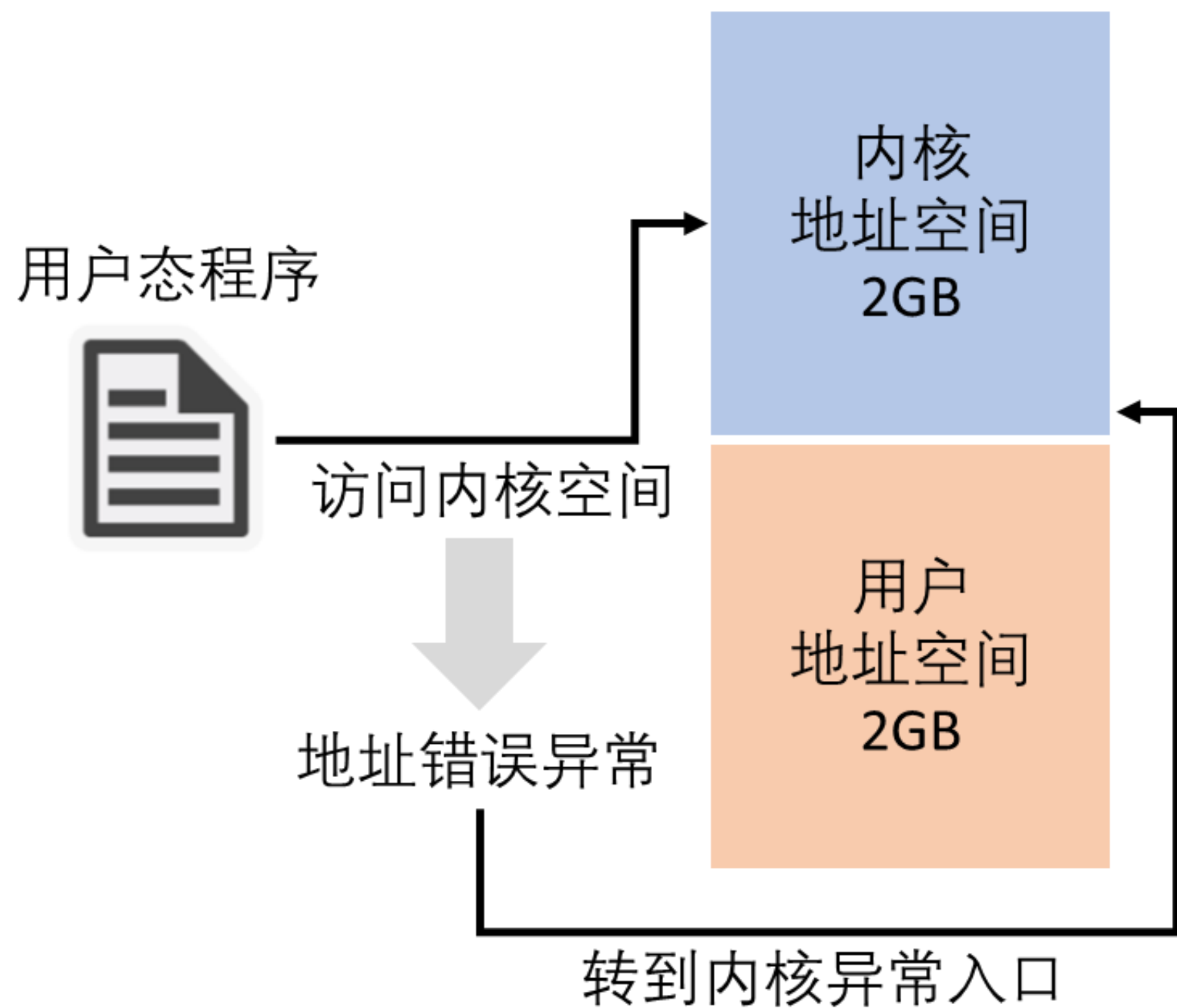
1. 虚拟内存与地址翻译
2. 进程结构
3. 进程调度

# 虚拟内存与地址翻译

- 虚拟地址空间隔离
- MIPS32地址翻译流程
- TLB操作

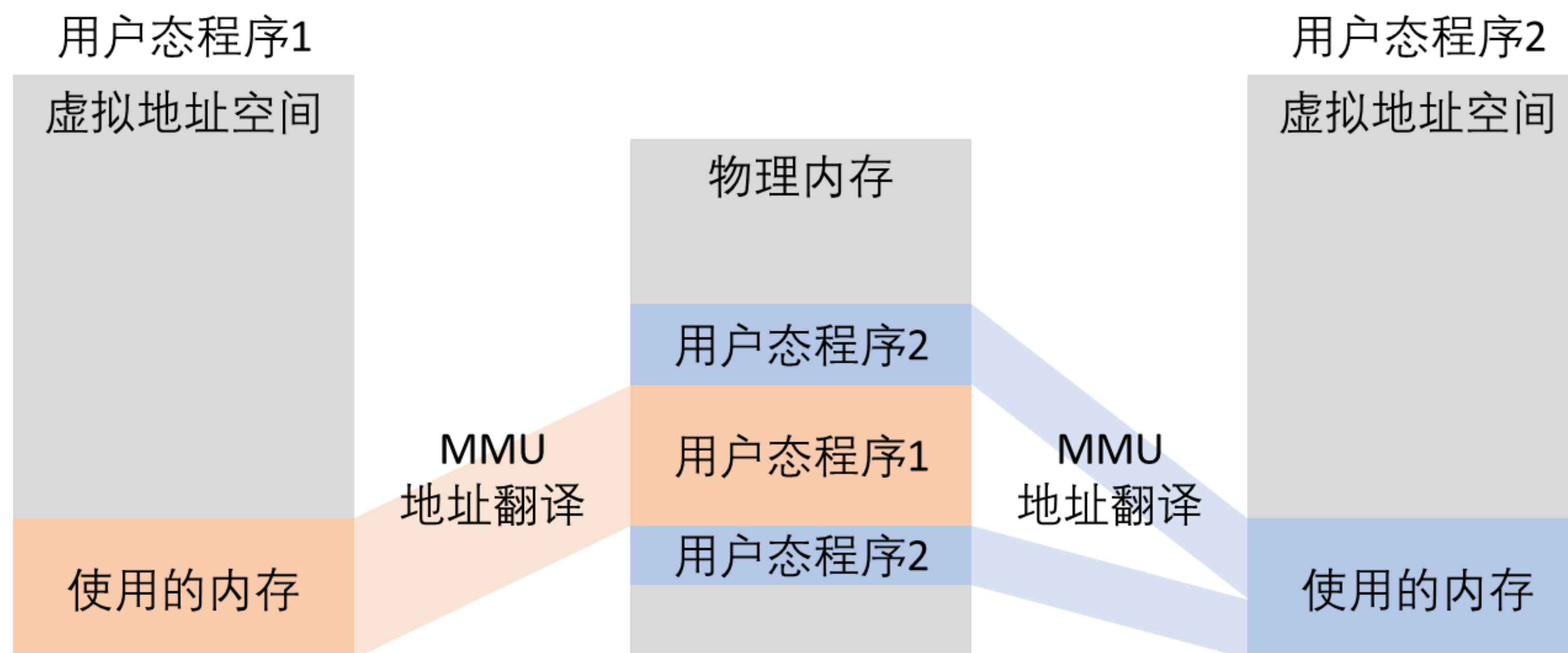
## 虚拟地址空间隔离

- 用户态程序不能访问内核地址空间：
  - 用户态访问内核地址空间时产生异常，从而进入内核态



## 虚拟地址空间隔离

- 用户态程序不能访问其他用户态程序的内存空间：
  - 通过MMU将不同用户态程序所使用的虚拟内存映射到不同的物理内存区域

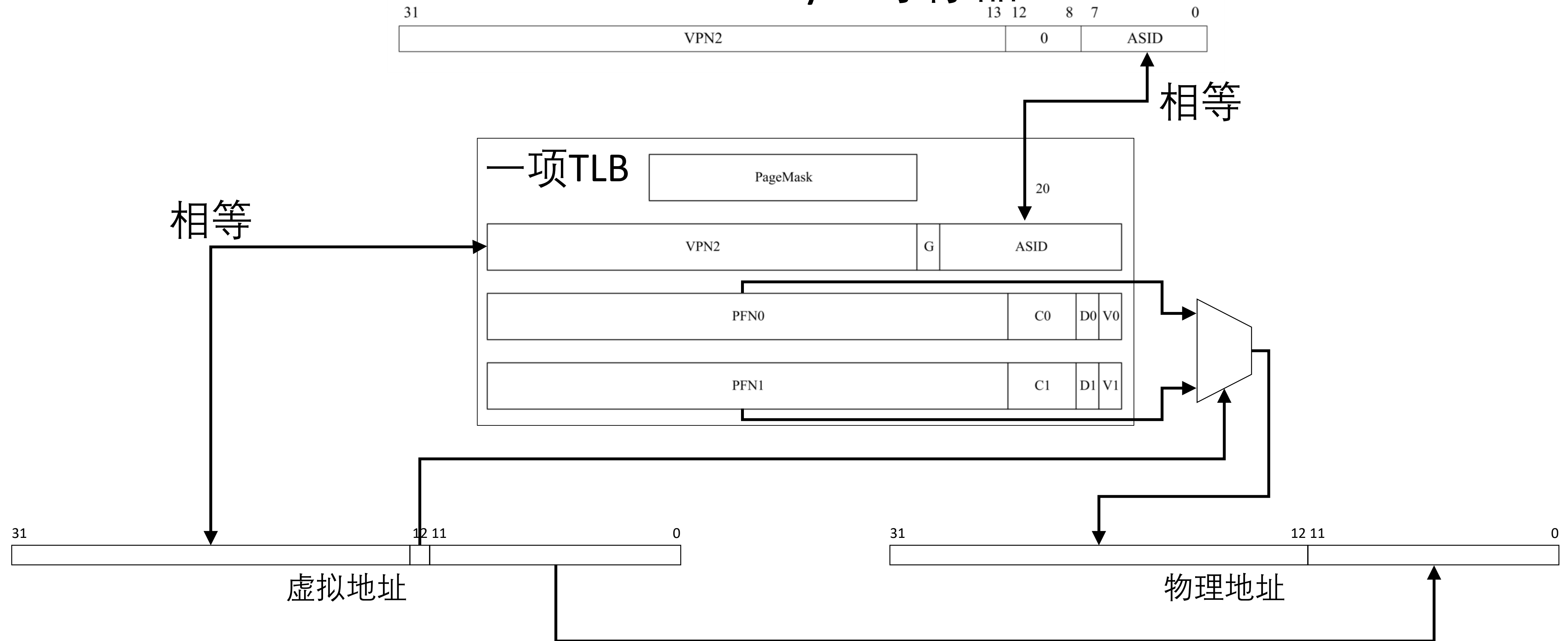


## MIPS32地址翻译流程

- MIPS32架构使用TLB进行虚拟地址翻译
- 页大小可变，最小4KB，最大256MB
- 一个TLB项映射两个物理页面
- TLB与COP0寄存器的ASID域需相等，操作系统内核进行进程调度时修改ASID，即可实现不同应用程序的地址空间隔离
- 以页大小为4KB为例，地址翻译流程如下：

# MIPS32地址翻译流程

# COP0 EntryHi寄存器



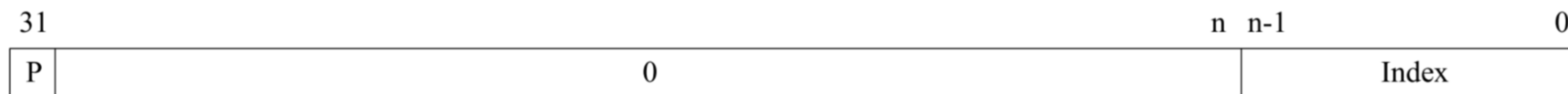
## TLB操作

- 与TLB的接口由7个COP0寄存器实现
  - 其中4个控制写入/读取的TLB数据
  - 3个控制写入/读取的TLB序号



## TLB操作

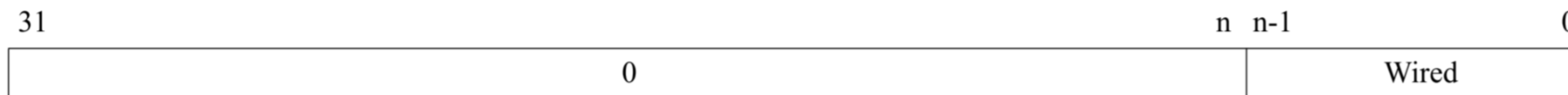
- Index寄存器：执行TLBWI指令时写该寄存器指向的TLB项，执行TLBR指令时读取该寄存器指向的TLB项



- Random寄存器：执行TLBWR指令时写该寄存器指向的TLB项，并随机更新

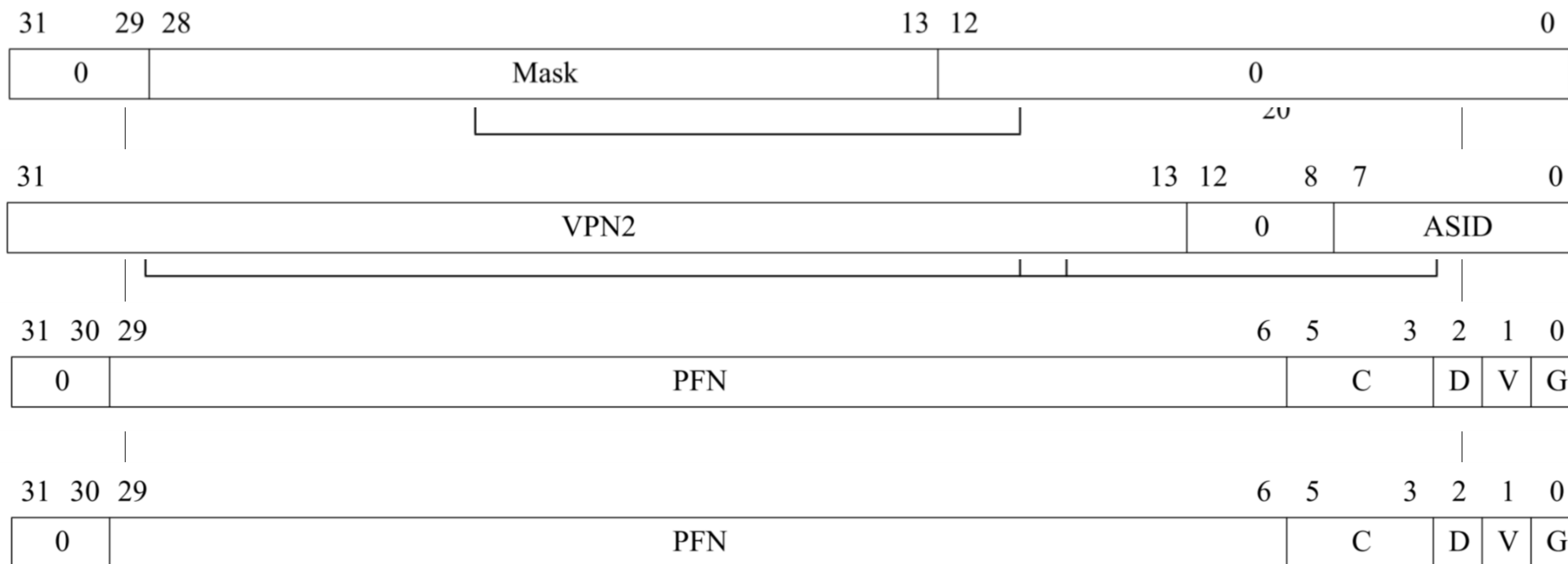


- Wired寄存器：Random寄存器的随机取值不会小于该寄存器的值



## TLB操作

- PageMask, EntryHi, EntryLo0, EntryLo1 :  
对应TLB的4个部分



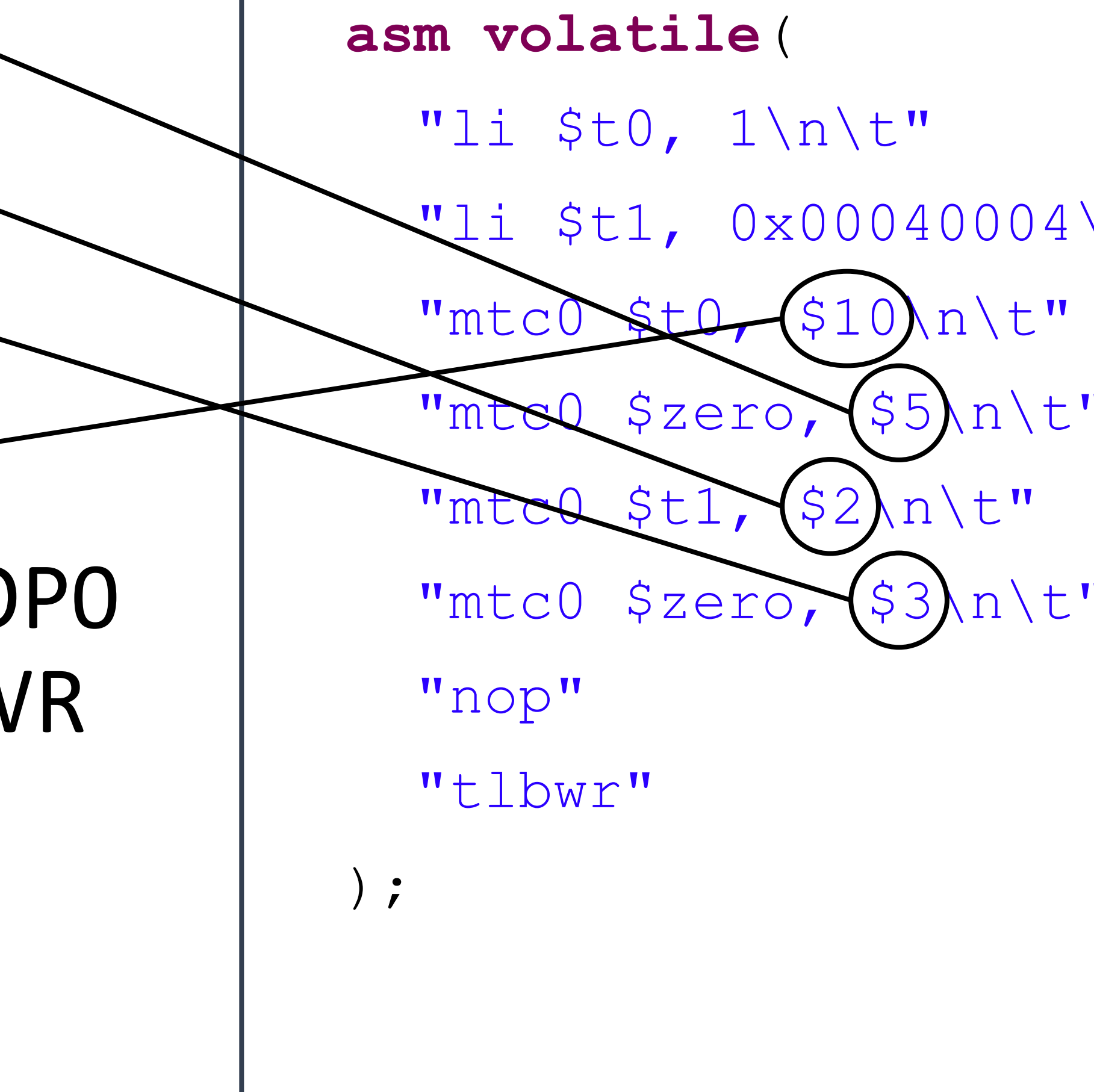
## TLB操作

- 例如：将虚拟地址0x0000 0000起始的一个页，映射到物理地址0x0100 0000起始的一个页，页大小为4KB
- EntryLo0[PFN]=0x01000, EntryLo0[V]=1
- EntryLo1[V]=0
- EntryHi[VPN2]=0
- 假设ASID为0x01

## TLB操作

- PageMask=0x0000 0000
- EntryLo0=0x0004 0004
- EntryLo1=0x0000 0000
- EntryHi=0x0000 0001
- 使用内联汇编设置各COP0寄存器，最后执行TLBWR指令随机写入一项TLB

```
asm volatile (  
    "li $t0, 1\n\t"  
    "li $t1, 0x00040004\n\t"  
    "mtc0 $t0, $10\n\t"  
    "mtc0 $zero, $5\n\t"  
    "mtc0 $t1, $2\n\t"  
    "mtc0 $zero, $3\n\t"  
    "nop"  
    "tlbwr"  
);
```





# 进程结构

- context
- task\_struct
- task\_union

## context

- 32个通用寄存器
- 异常返回地址EPC
- 顺序很重要
  - 与异常入口保存顺序相同
  - 通过struct context\*指针访问保存在内核栈上的内容

```
typedef struct {  
    unsigned int epc;  
    unsigned int at;  
    unsigned int v0, v1;  
    unsigned int a0, a1, a2, a3;  
    unsigned int t0, t1, t2, t3, t4, t5, t6, t7;  
    unsigned int s0, s1, s2, s3, s4, s5, s6, s7;  
    unsigned int t8, t9;  
    unsigned int hi, lo;  
    unsigned int gp;  
    unsigned int sp;  
    unsigned int fp;  
    unsigned int ra;  
} context;
```

## task\_struct

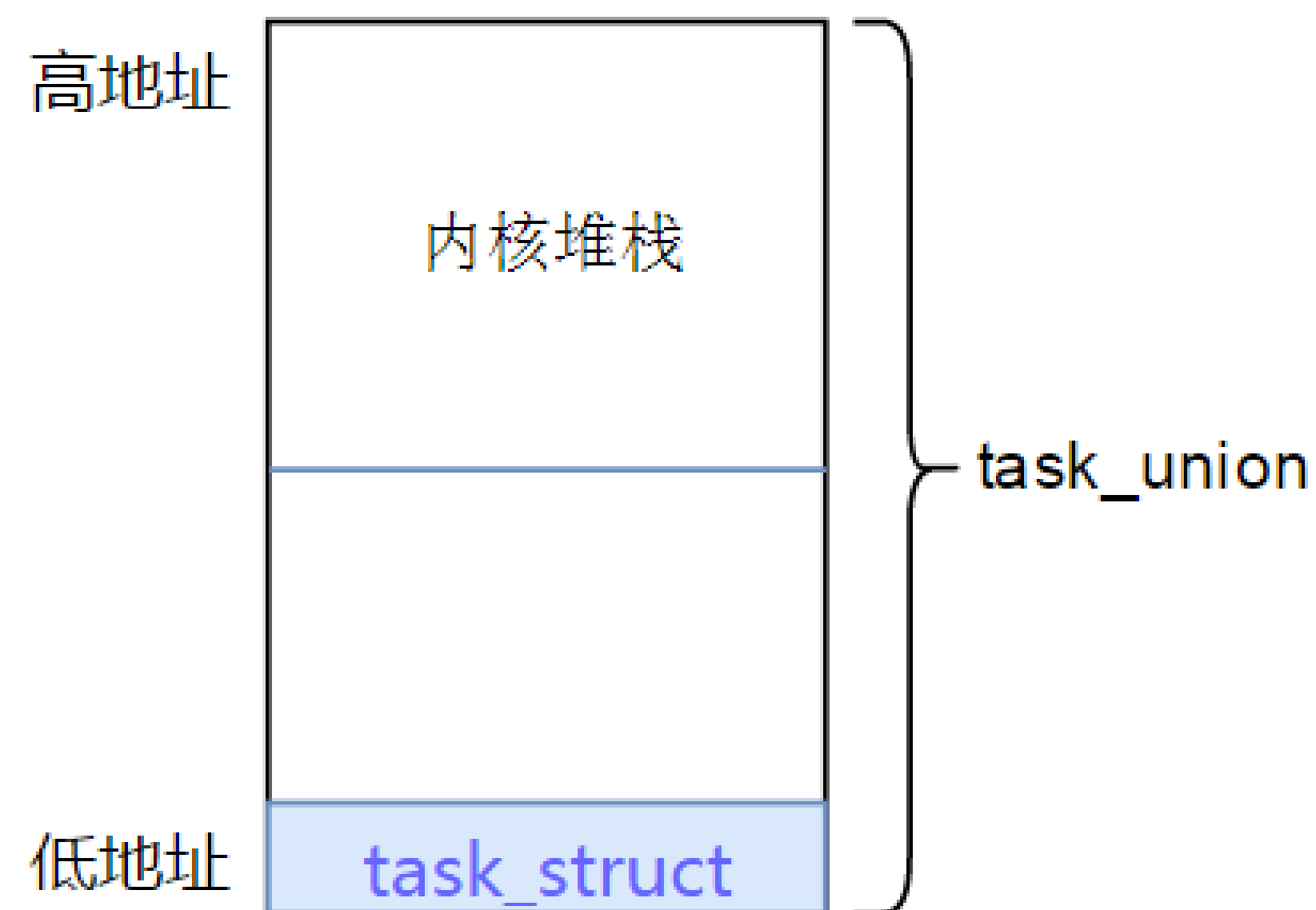
- context 上下文
- ASID 进程号
- counter 剩余时间片
- name 进程名称
- start\_time 创建时间

```
typedef struct {  
    context context;  
    int ASID;  
    unsigned int counter;  
    char name[32];  
    unsigned long start_time;  
} task_struct;
```

## task\_union

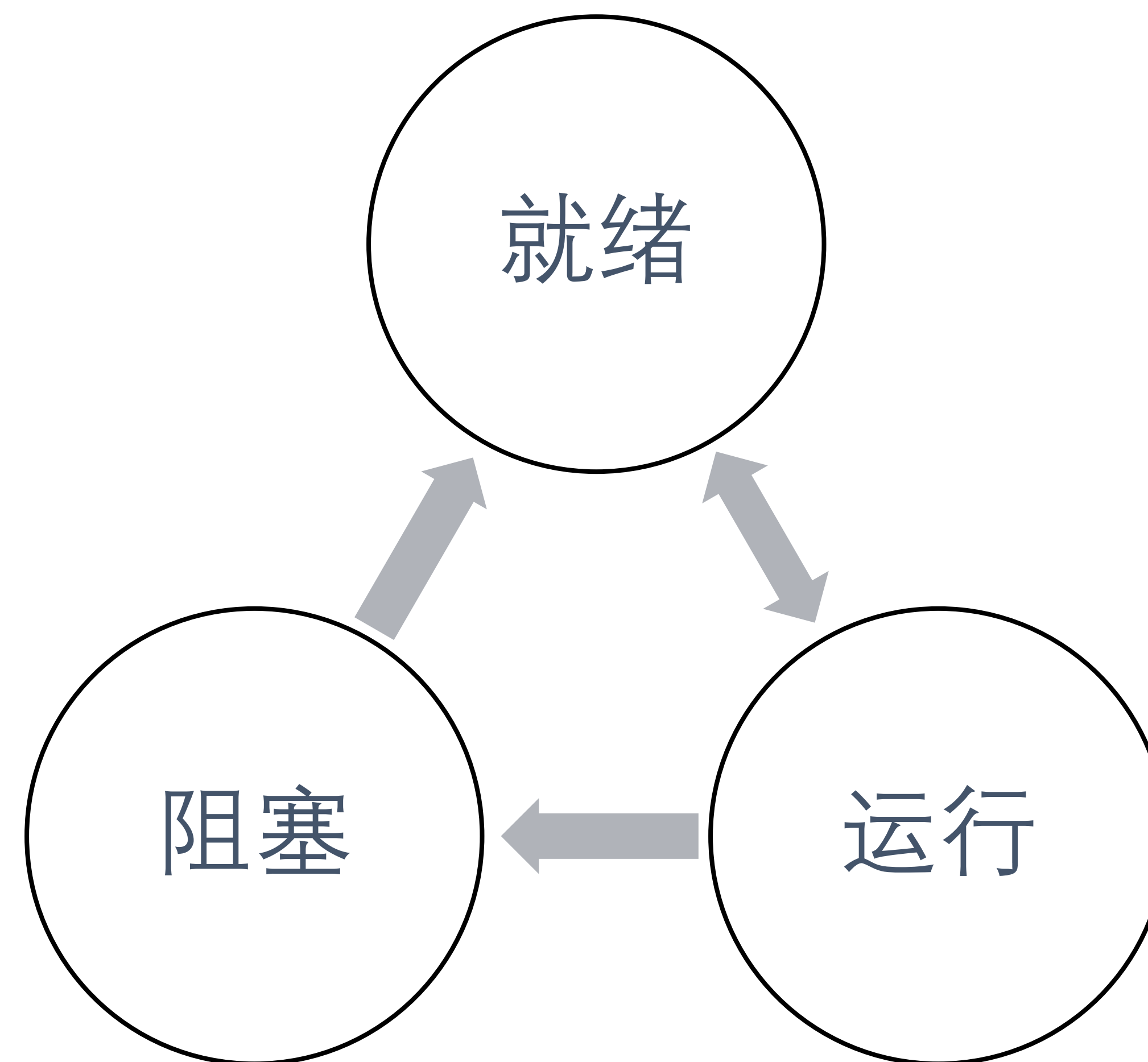
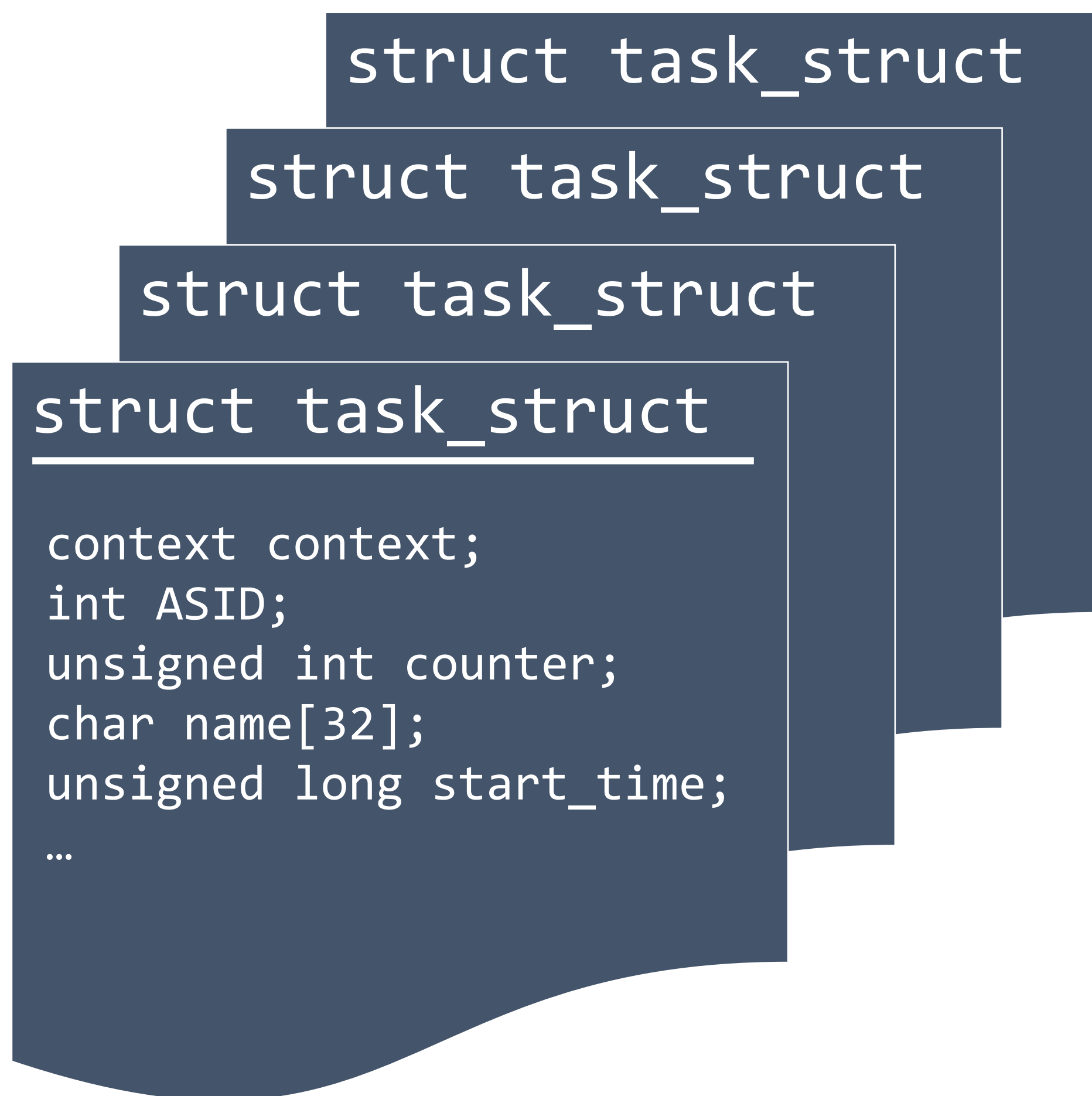
- 进程的内核态堆栈(大小一个页)与task\_struct共用task\_union
- task\_struct位于内核态堆栈的底部

```
typedef union {  
    task_struct task;  
    unsigned char kernel_stack[4096];  
} task_union;
```





## 多进程结构

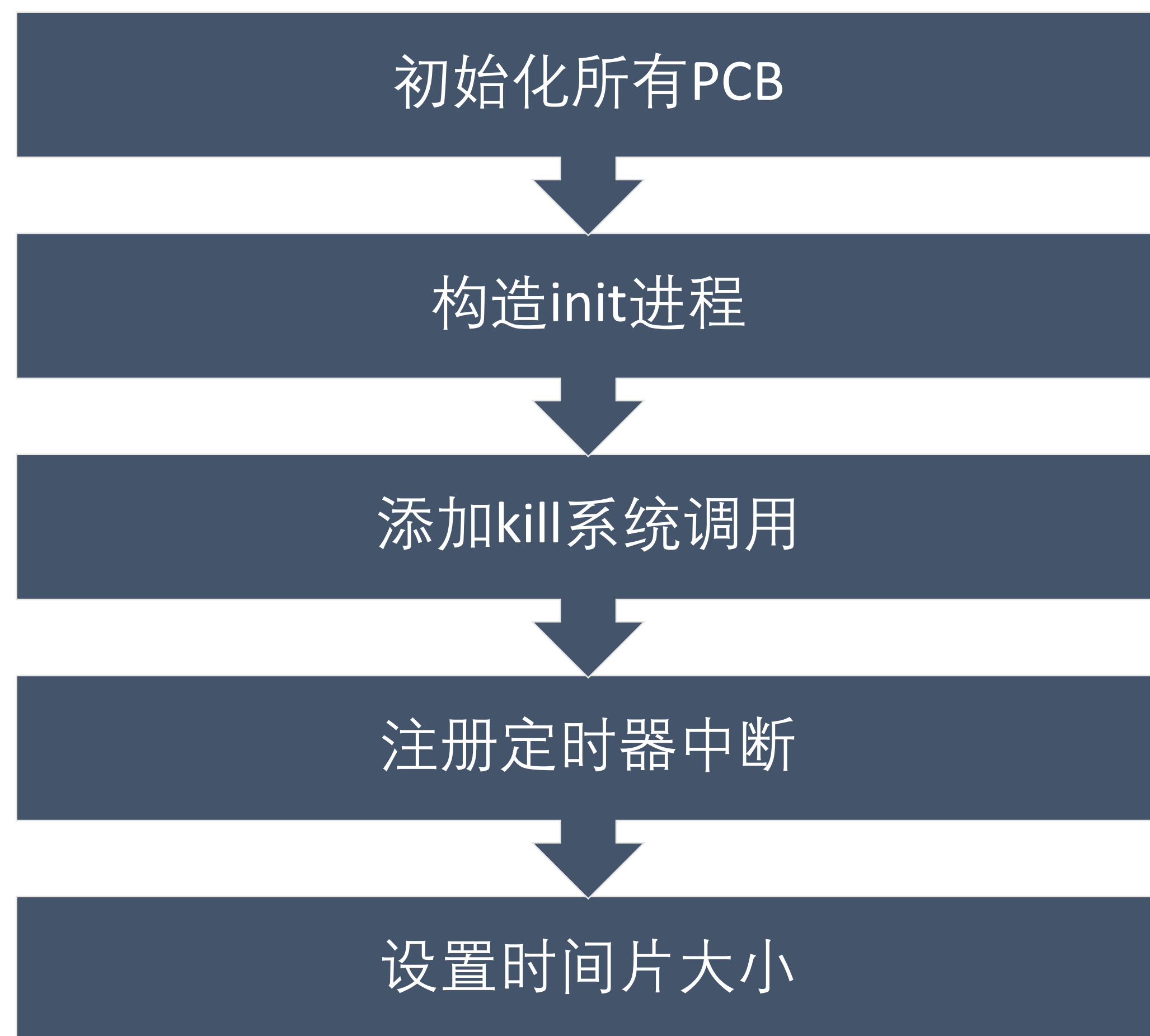


# 进程调度

- 进程管理初始化
- 进程调度
- 进程创建与删除



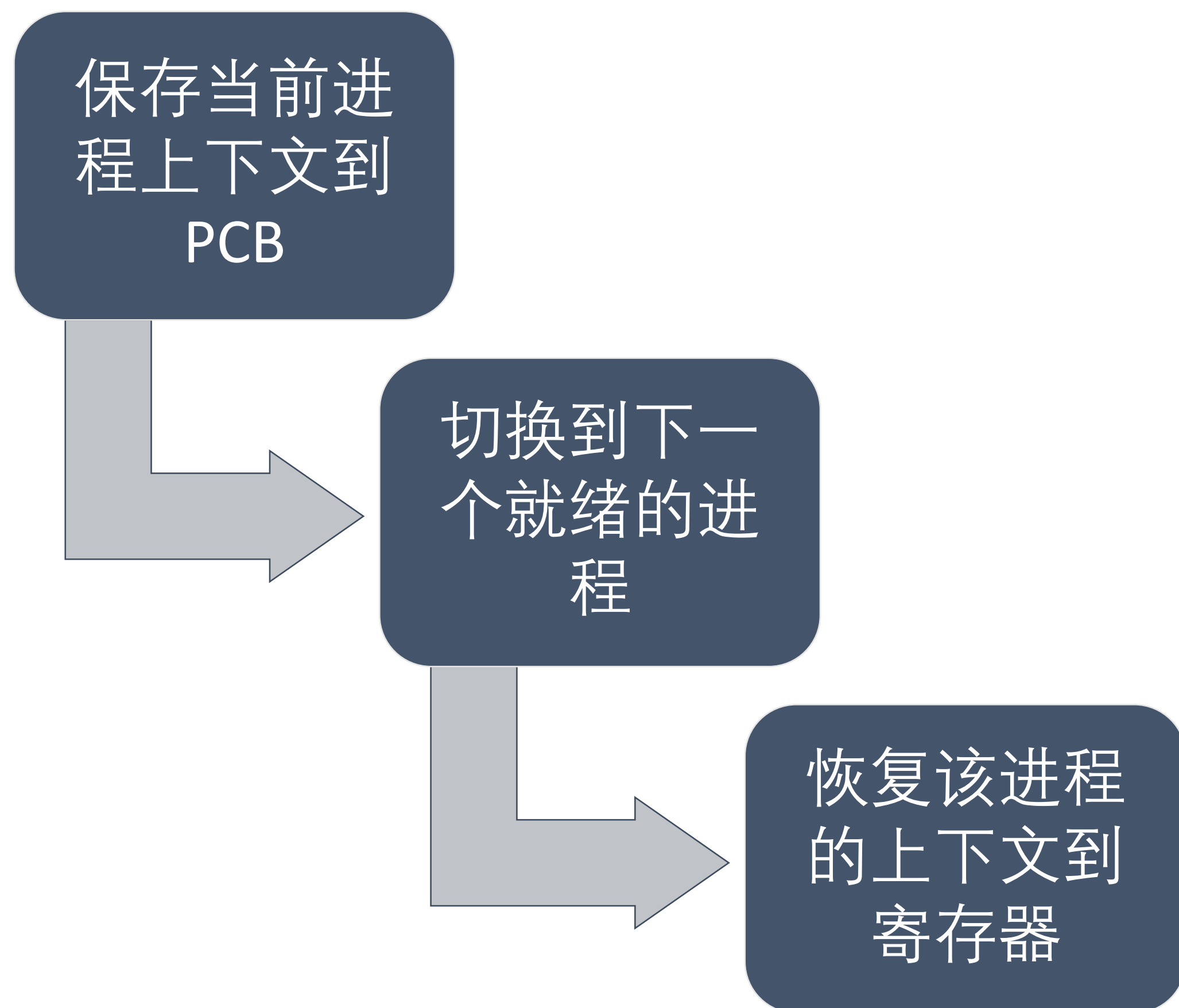
## 进程管理初始化



```
void init_pc() {
    int i;
    for (i = 1; i < 8; i++)
        pcb[i].ASID = -1;
    pcb[0].ASID = 0;
    pcb[0].counter = PROC_DEFAULT_TIMESLOTS;
    kernel_strcpy(pcb[0].name, "init");
    curr_proc = 0;
    register_syscall(10, pc_kill_syscall);
    register_interrupt_handler(7, pc_schedule);

    asm volatile(
        "li $v0, 1000000\n\t"
        "mtc0 $v0, $11\n\t"
        "mtc0 $zero, $9");
}
```

# 进程调度



```
void pc_schedule(unsigned int status, unsigned int cause, context* pt_context) {  
    // Save context  
    copy_context(pt_context, &(pcb[curr_proc].context));  
    int i;  
    for (i = 0; i < 8; i++) {  
        curr_proc = (curr_proc + 1) & 7;  
        if (pcb[curr_proc].ASID >= 0)  
            break;  
    }  
    if (i == 8) {  
        kernel_puts("Error: PCB[0] is invalid!\n", 0xffff, 0);  
        while (1)  
            ;  
    }  
    // Load context  
    copy_context(&(pcb[curr_proc].context), pt_context);  
    asm volatile("mtc0 $zero, $9\n\t");  
}
```



# 进程调度



## 进程创建与删除

- 将一些信息写入分配好的PCB
- 设置代码段，堆栈，静态数据段的信息

```
void pc_create(int asid, void (*func)(), unsigned int init_sp,  
               unsigned int init_gp, char* name)  
{  
    pcb[asid].context.epc = (unsigned int)func;  
    pcb[asid].context.sp = init_sp;  
    pcb[asid].context.gp = init_gp;  
    kernel_strcpy(pcb[asid].name, name);  
    pcb[asid].ASID = asid;  
}
```

## 进程创建与删除

- 创建
  - 将信息写入分配的PCB
  - 代码段、静态数据段、堆栈
  - 进程名，进程号

```
void pc_create(int asid, void (*func)(), unsigned int init_sp,  
               unsigned int init_gp, char* name)  
{  
    pcb[asid].context.epc = (unsigned int)func;  
    pcb[asid].context.sp = init_sp;  
    pcb[asid].context.gp = init_gp;  
    kernel_strcpy(pcb[asid].name, name);  
    pcb[asid].ASID = asid;  
}
```

## 进程创建与删除

- 删除
  - 将当前进程的PCB设置为无效
  - 无法删除init进程

```
int pc_kill(int proc) {  
    proc &= 7;  
    if (proc != 0 && pcb[proc].ASID >= 0) {  
        pcb[proc].ASID = -1;  
        return 0;  
    } else if (proc == 0)  
        return 1;  
    else  
        return 2;  
}
```





THANK YOU