习题课一

并发进程

方　钰

主要知识点：

操作系统的一般原理：

1. 进程的基本概念

2. 进程调度算法与死锁的概念

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

UNIX操作系统：

4. UNIX进程的两个执行状态及进程图象

5. UNIX中断的详细处理过程

主要知识点：

操作系统的一般原理：

1. 进程的基本概念
2. 进程调度算法与死锁的概念
3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

UNIX操作系统：

4. UNIX进程的两个执行状态及进程图象
5. UNIX中断的详细处理过程

主要知识点：

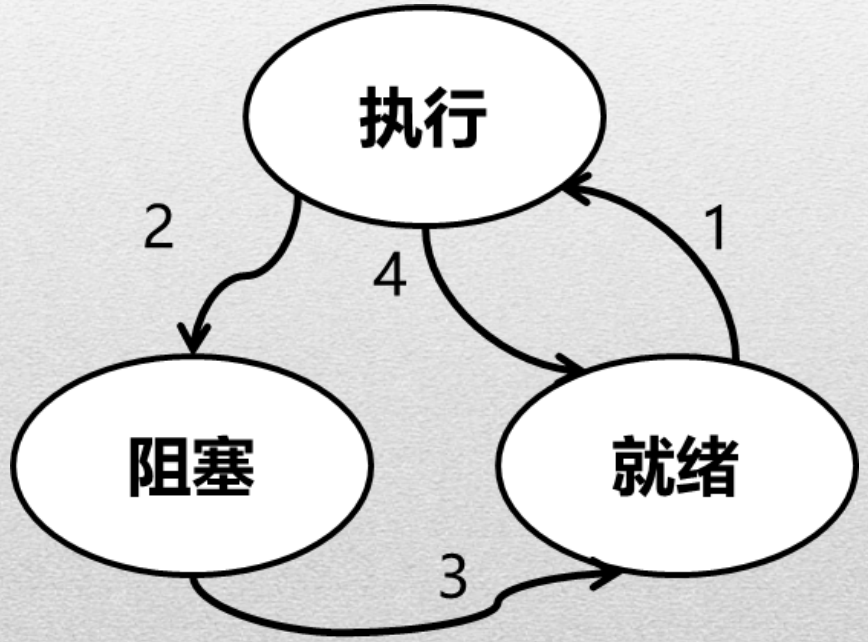
1. 进程的基本概念

进程与程序的主要区别

程序 → 进程

程序	进程
一组数据与指令代码的集合	**结构特征** 代码段、数据段、堆栈段、进程控制块
静态的 存放在某种介质上	**动态性**，具有生命周期 “由创建而产生，由调度而执行，由撤销而消亡”

进程的基本调度状态



（1）下列选项中，可能将进程唤醒的事件是（　　）：

Ⅰ. I/O结束　　Ⅱ. 某进程入睡　　Ⅲ. 当前进程的时间片用完

A. 仅Ⅰ　　B. 仅Ⅲ　　C. 仅Ⅰ、Ⅱ　　D. Ⅰ、Ⅱ、Ⅲ

（2）某进程所要求的一次打印输出结束，该进程被_____，进程的状态将从_____。

A. 阻塞　　B. 执行　　C. 唤醒　　D. 运行到阻塞

E. 就绪到运行　　F. 阻塞到就绪　　H. 运行到就绪

掌握所有状态的特征及状态之间变化发生的条件和结果

（3） _____可能会引起处理机从一个进程转到另一个进程。

A. 一个进程从运行状态变为等待状态

B. 一个进程从运行状态变为就绪状态

C. 一个就绪状态进程的优先级降低

D. 一个进程运行完成而撤离系统

（4）设系统中有n（n>2）个进程，且当前不在执行进程调度程序，试考虑下述4种情况：

A. 没有执行进程，有2个就绪进程，n-2个进程处于等待状态；

B. 有1个执行进程，没有就绪进程，n-1进程处于等待状态；

C. 有1个执行进程，有1个就绪进程，n-2进程处于等待状态；

D. 没有执行进程，没有就绪进程，n个进程处于等待状态。

上述情况中，不可能发生的情况是_____ 。

主要知识点：

操作系统的一般原理：

1. 进程的基本概念

2. 进程调度算法与死锁的概念

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

UNIX操作系统：

4. UNIX进程的两个执行状态及进程图象

5. UNIX中断的详细处理过程

主要知识点：

2. 进程调度算法与死锁的概念

处理机调度概念与常用的调度算法：FIFO，短作业，优先权，时间片（流程、指标）
银行家算法

（1）某系统正在执行三个进程P1、P2和P3，各进程的计算（CPU）时间和I/O时间比例如下表所示：

进程	计算时间	I/O时间
P1	90%	10%
P2	50%	50%
P3	15%	85%

为提高系统资源利用率，合理的进程优先级设置应为 _____ 。

（2）某系统采用基于优先权的非抢占式进程调度，完成一次进程调度和进程切换的系统时间开销为1μs。T时刻就绪队列中有3个进程P1、P2和P3，在就绪队列中的等待时间、需要的CPU时间和优先权如下表所示：

进程	已等待时间	需要的CPU时间	优先权
P1	30μs	12μs	10
P2	15μs	24μs	30
P3	18μs	36μs	20

从T时刻起系统开始进程调度，则系统的平均周转时间为 _____ 。

（3）假设4个作业到达系统的时刻和运行时间如下表所示：

作业	到达时刻t	运行时间
J1	0	3
J2	1	3
J3	1	2
J4	3	1

系统在t=2时开始作业调度。若分别采用先来先服务和短作业优先调度算法，则选中的作业分别是为 _____ 。

主要知识点：

操作系统的一般原理：

1. 进程的基本概念

2. 进程调度算法与死锁的概念

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

UNIX操作系统：

4. UNIX进程的两个执行状态及进程图象

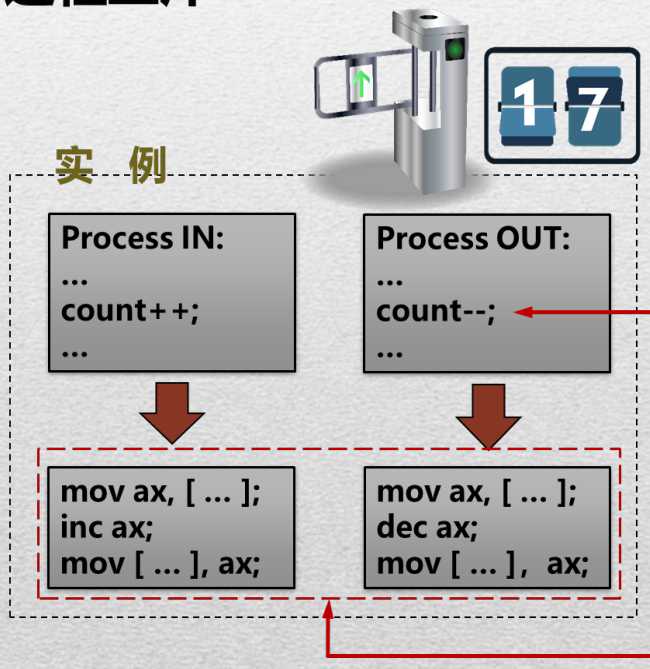
5. UNIX中断的详细处理过程

主要知识点：

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

同步与互斥、信号量的含义、P，V操作、经典的进程通信问题

进程互斥



临界资源：同时只允许一个进程使用的软件和硬件资源。

临界区：访问临界资源的程序段。

相关临界区：访问同一临界资源的临界区。

进程同步

协同工作的几个进程需要在某些确定的点上协调他们的工作。

一个进程达到了一个确定的点（同步点）后，除非另一个进程已经完成了某些操作，否则就不得不停下来以等待这些操作执行结束。

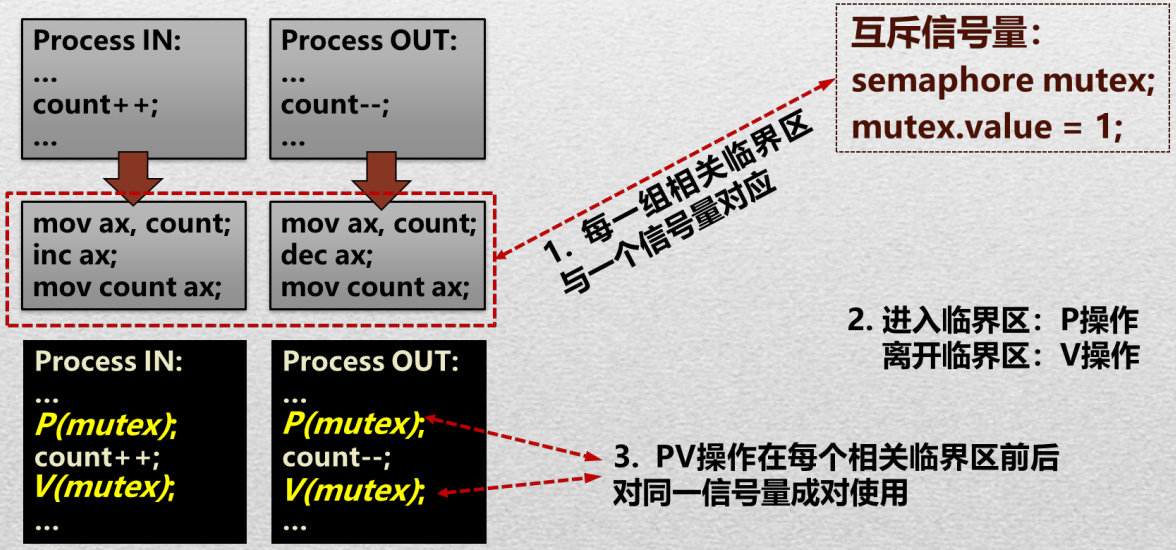
主要知识点：

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

同步与互斥、信号量的含义、P，V操作、经典的进程通信问题

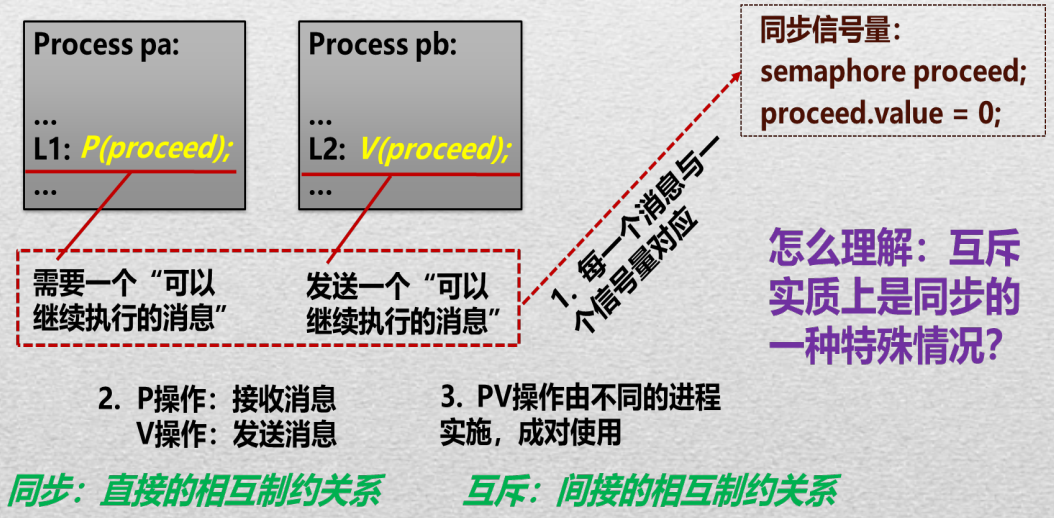
利用“信号量”实现进程互斥

互斥：任意时刻只能有一个进程使用该资源



利用“信号量”实现进程同步

同步：在同步点上等待“可以继续执行”的消息



1. 有m个进程共享同一临界资源，若使用信号量机制实现对临界资源的互斥访问，则信号量值的变化范围是 _____ 。

2. 假设一个信号量初值为3，当前值为1。M表示初始条件下该资源的可用个数，N表示等待该资源的进程数，则M和N分别是（ ）
   A. 3，0　　B. 1，0　　C. 1，2　　D. 2，0

3. P、V操作必须在屏蔽中断下执行，这种不可被中断的过程称为_____。
   A. 初始化程序　　B. 原语　　C. 子程序　　D. 控制模块

主要知识点：

操作系统的一般原理：

1. 进程的基本概念

2. 进程调度算法与死锁的概念

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

UNIX操作系统：

4. UNIX进程的两个执行状态及进程图象

5. UNIX中断的详细处理过程

主要知识点：

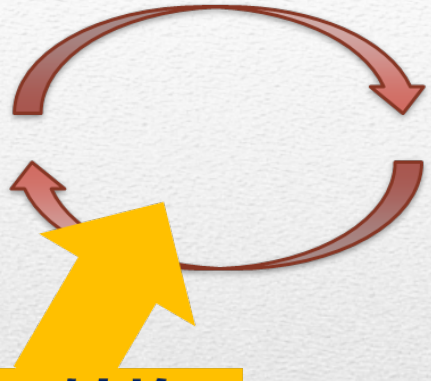
4. UNIX进程的两个执行状态及进程图象

用户态 User Mode

执行用户程序，提供用户功能

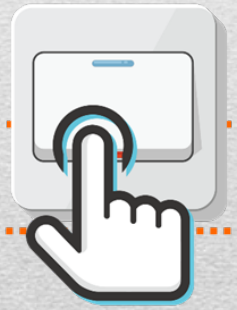
核心态 Kernel Mode

执行内核程序，提供系统功能



可以在一定时机相互转换

内核不是与用户进程平行运行的孤立的进程集合。



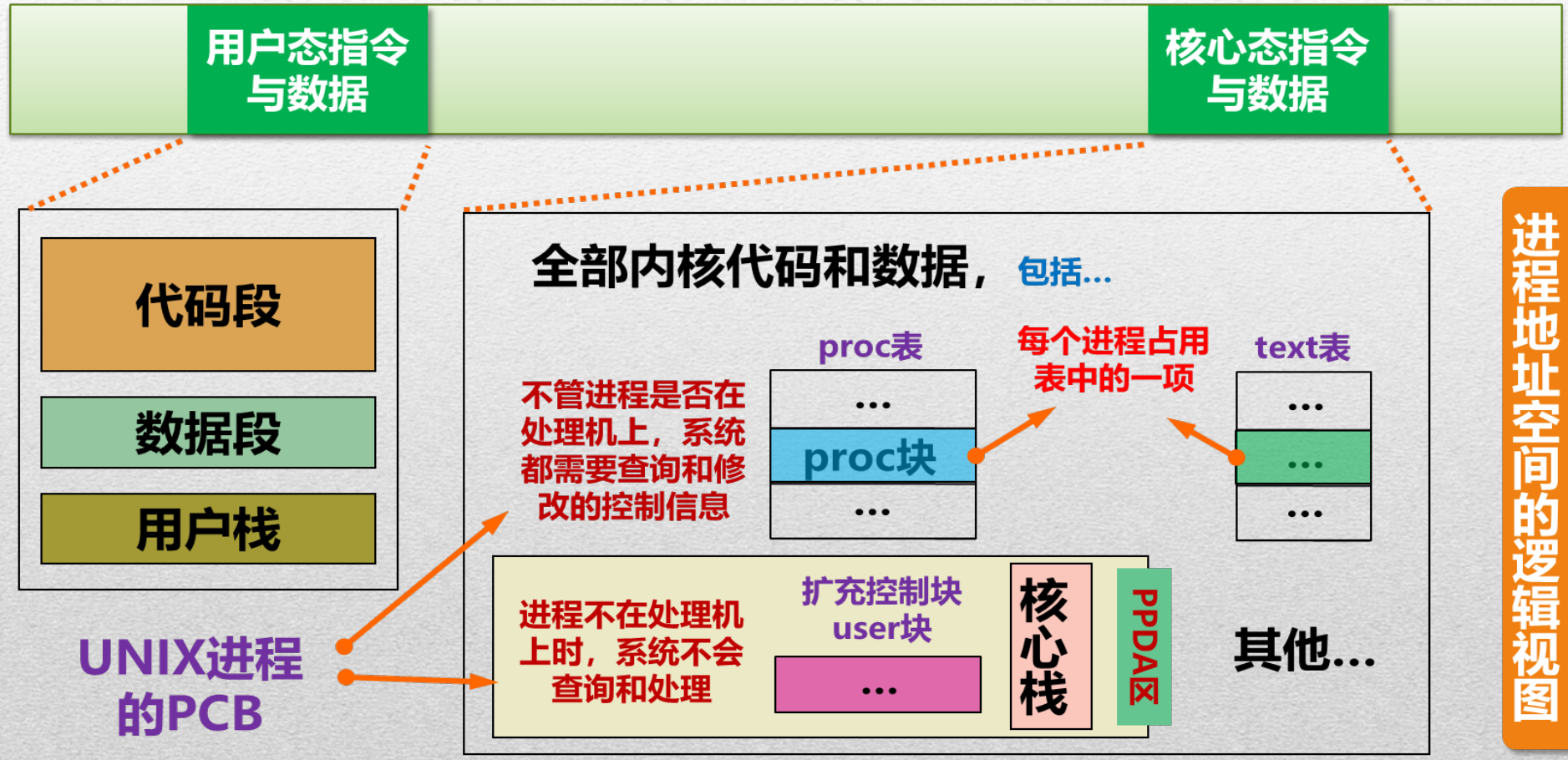
中断、异常、系统调用

在核心态下执行内核代码的进程完成了内核功能！

进程的两个执行状态

主要知识点：

4. UNIX进程的两个执行状态及进程图象



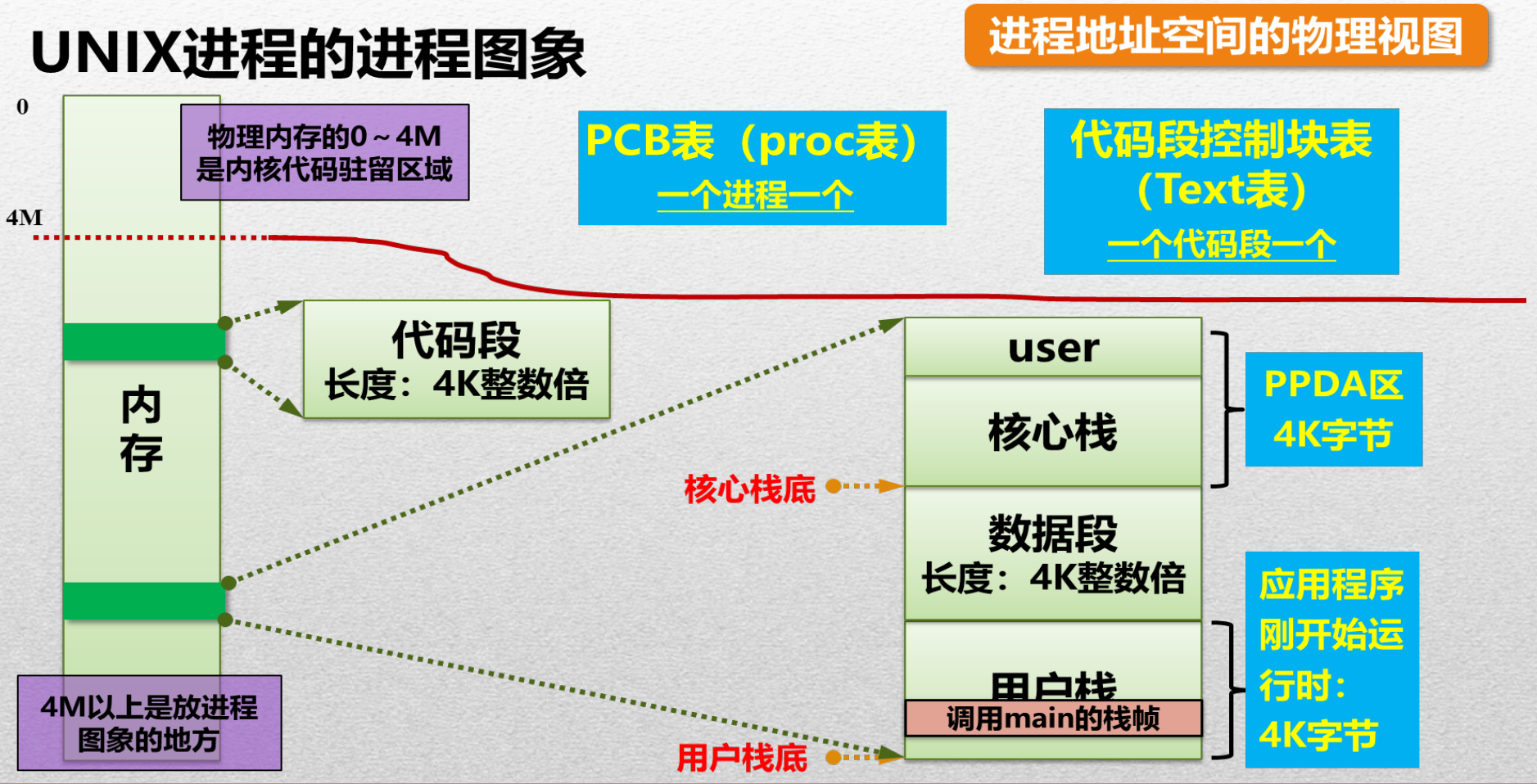
进程图象的构成

特别是用户栈和核心栈的构成

主要知识点：

4. UNIX进程的两个执行状态及进程图象

UNIX进程的进程图象

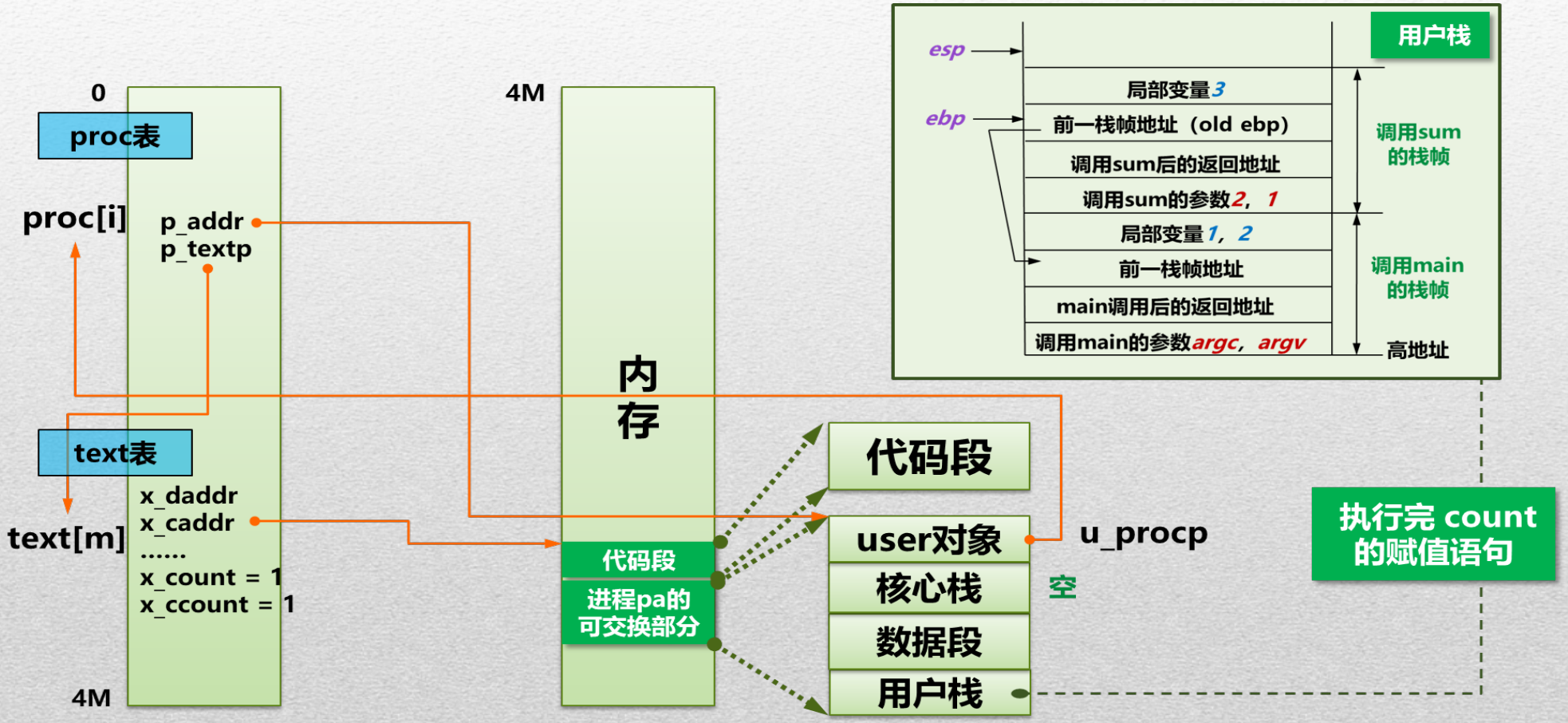


进程图象的构成

特别是用户栈和核心栈的构成

主要知识点：

4. UNIX进程的两个执行状态及进程图象



进程图象的构成

（1）当计算机区分了核心态和用户态指令之后，用户态到核心态的转换是由（　　）完成的。

A. 硬件　　B. 核心态程序　　C. 用户程序　　D. 中断处理程序

（2）假定下列指令已经装入指令寄存器。则执行时不可能导致CPU从用户态进入核心态的是（　　）。

A. DIV R0, R1　　B. INT n

C. NOT R0　　D. MOV R0, addr

主要知识点：

操作系统的一般原理：

1. 进程的基本概念

2. 进程调度算法与死锁的概念

3. 进程通信（临界区、临界资源；进程同步与互斥；信号量）

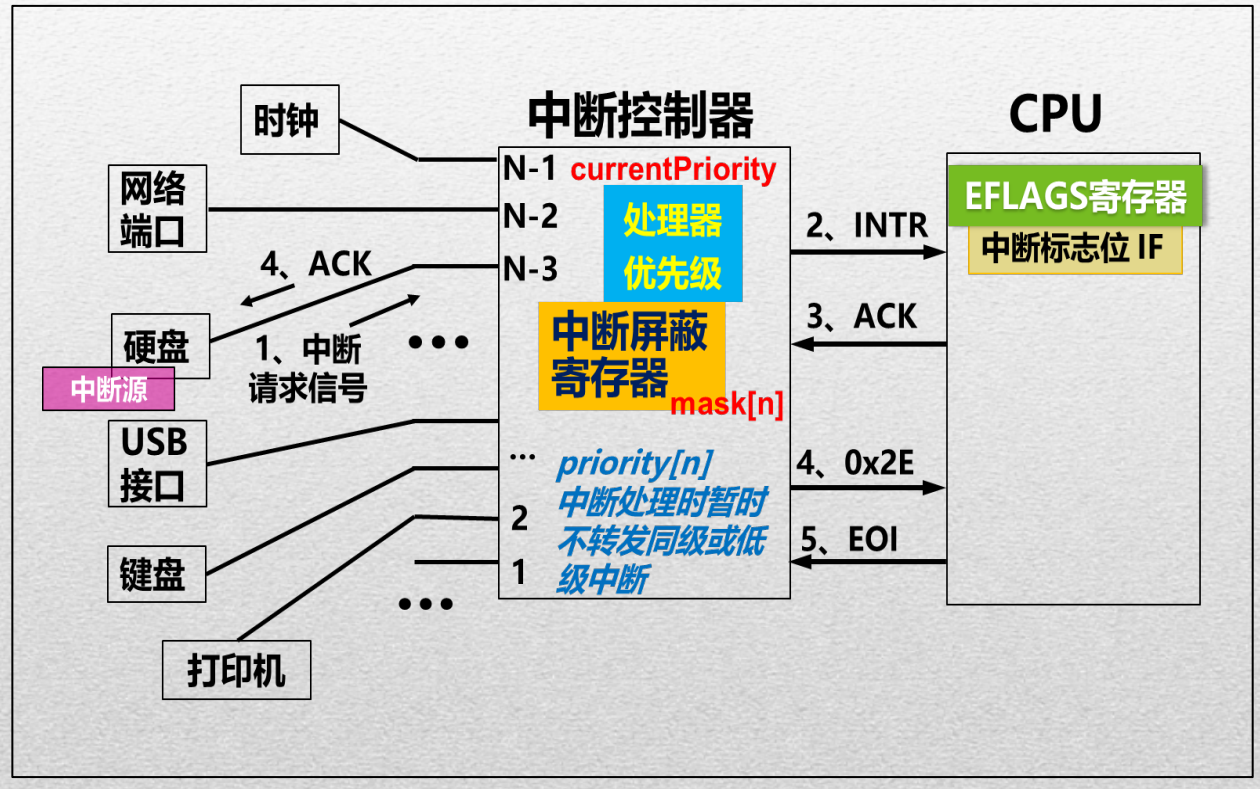
UNIX操作系统：

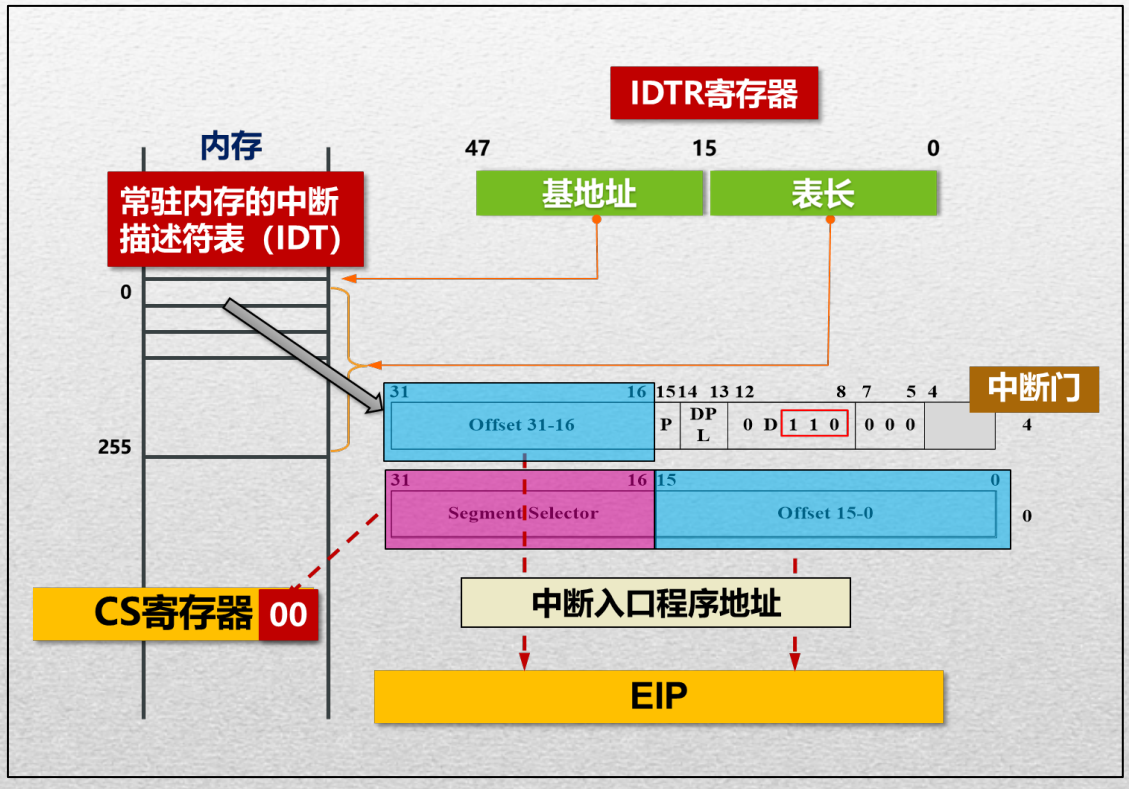
4. UNIX进程的两个执行状态及进程图象

5. UNIX中断的详细处理过程

主要知识点：

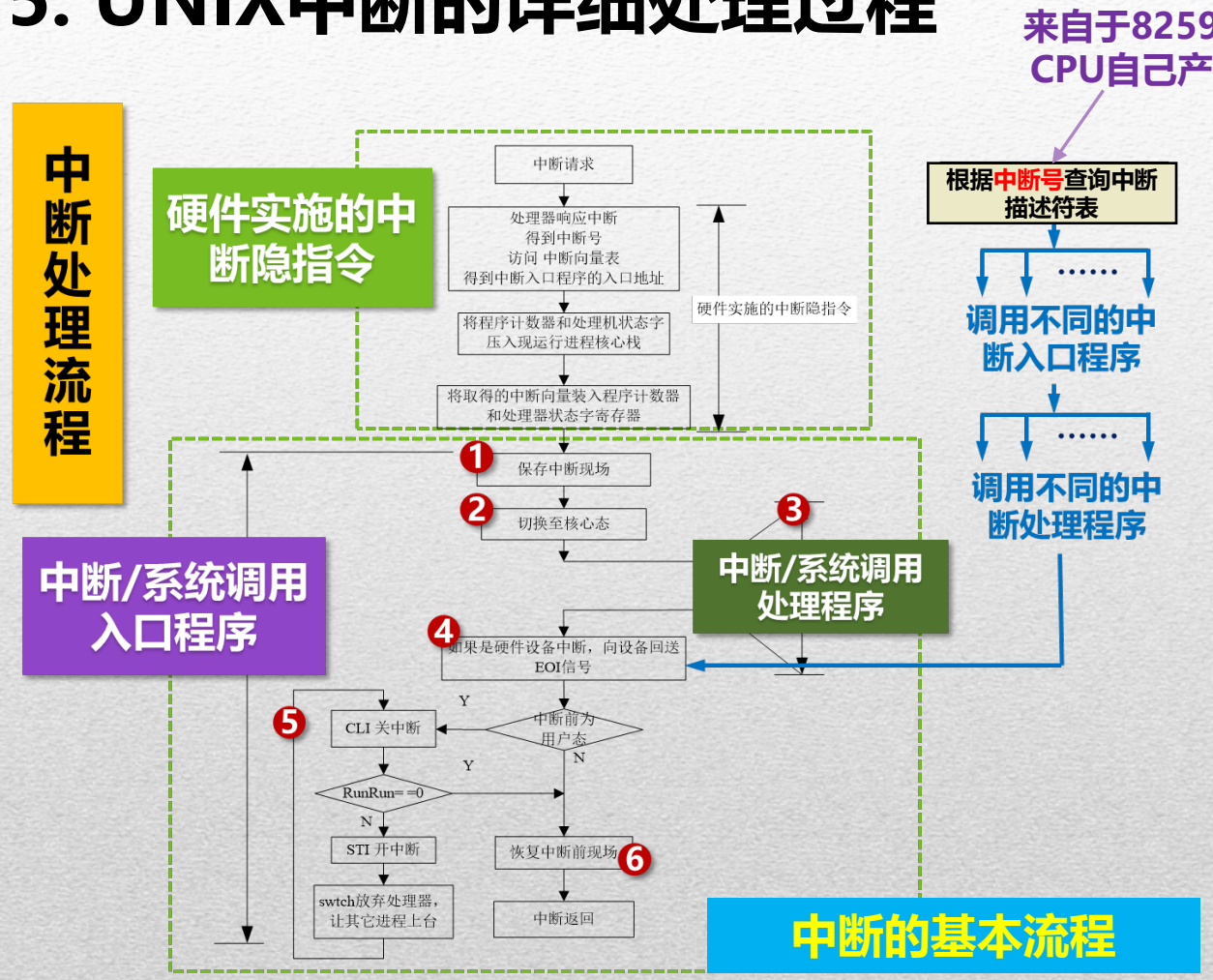
5. UNIX中断的详细处理过程

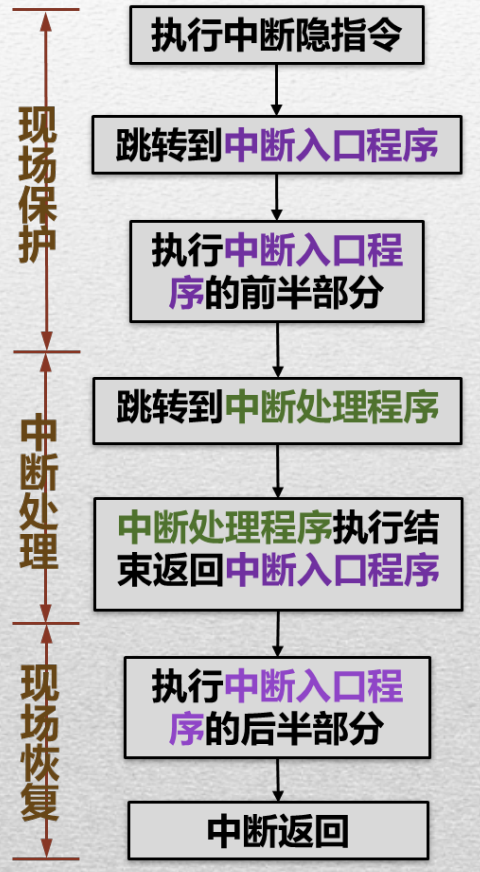


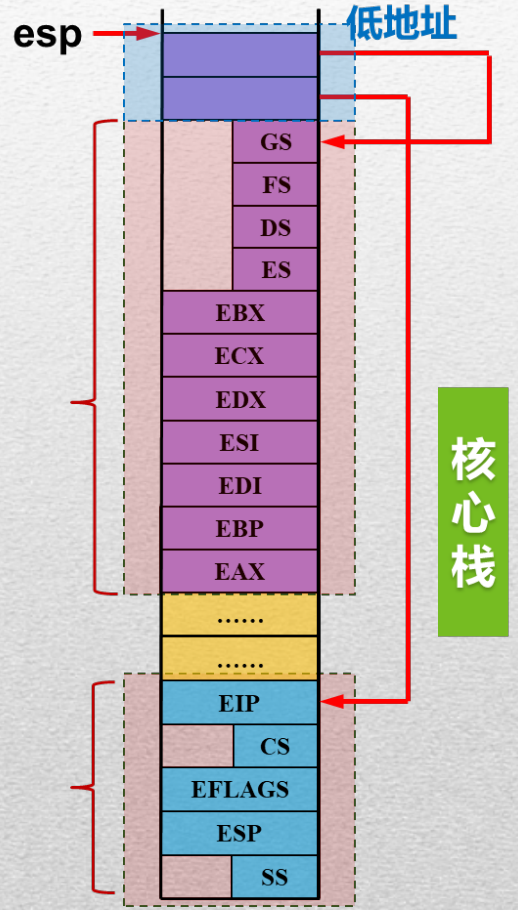


主要知识点：

5. UNIX中断的详细处理过程

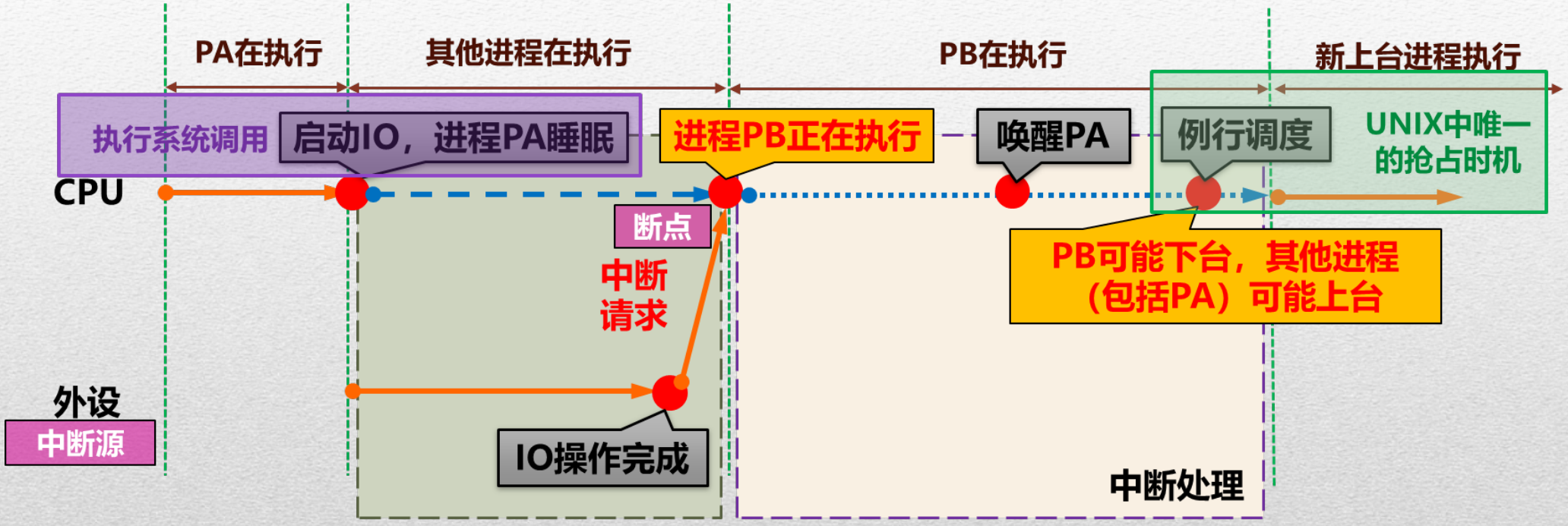






主要知识点：

5. UNIX中断的详细处理过程



1. 保证了CPU和设备之间的并行操作

2. 提供了进程执行内核代码的机会

3. 多道程序并发的硬件基础

主要知识点：

现运行进程

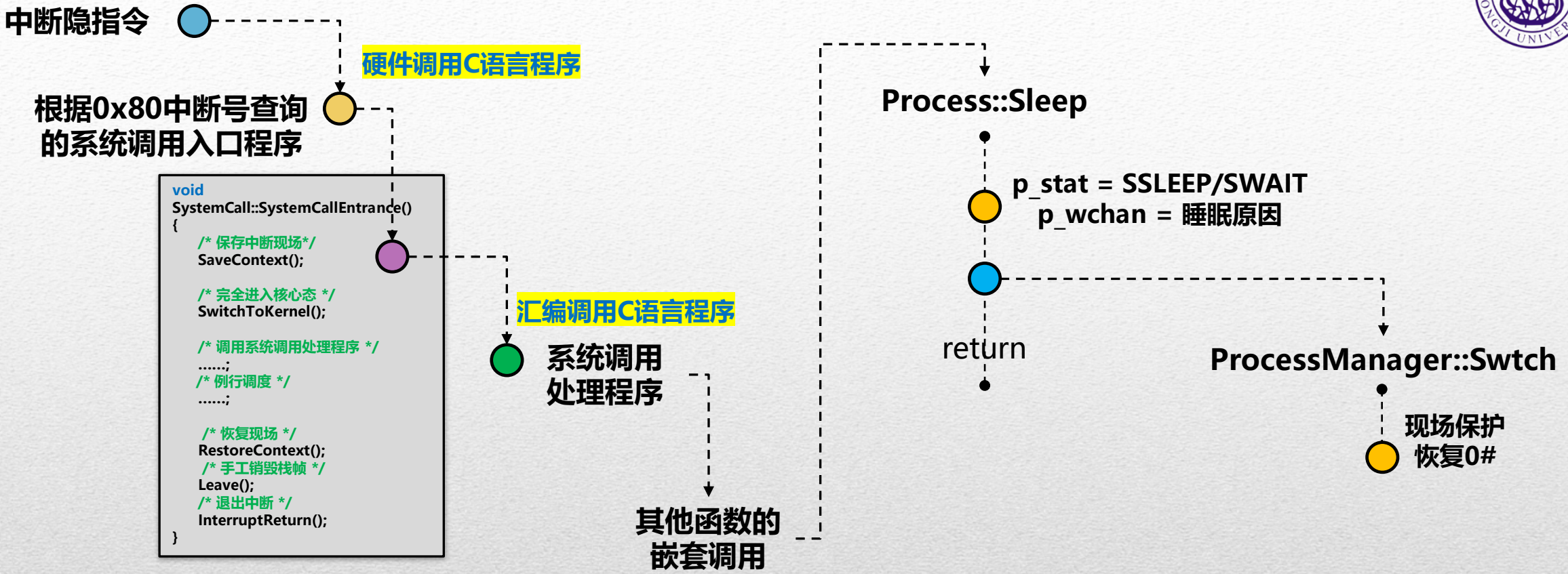
p_stat = SRUN;
p_flag有SLOAD

主要知识点：

现运行进程

p_stat = SRUN;
p_flag有SLOAD

如果进程在执行过程中需要使用系统资源


中断隐指令
硬件调用C语言程序
根据0x80中断号查询
的系统调用入口程序
void
SystemCall::SystemCallEntrance()
{
/* 保存中断现场*/
SaveContext();
/* 完全进入核心态 */
SwitchToKernel();
/* 调用系统调用处理程序 */
……;
/* 例行调度 */
……;
/* 恢复现场 */
RestoreContext();
/* 手工销毁栈帧 */
Leave();
/* 退出中断 */
InterruptReturn();
}
汇编调用C语言程序
系统调用
处理程序
其他函数的
嵌套调用
Process::Sleep
p_stat = SSLEEP/SWAIT
p_wchan = 睡眠原因
return
ProcessManager::Swtch
现场保护
恢复0#

中断隐指令

硬件调用C语言程序

根据0x80中断号查询
的系统调用入口程序

```
void
SystemCall::SystemCallEntrance()
{
    /* 保存中断现场*/
    SaveContext();

    /* 完全进入核心态 */
    SwitchToKernel();

    /* 调用系统调用处理程序 */
    ……;
    /* 例行调度 */
    ……;

     /* 恢复现场 */
    RestoreContext();
     /* 手工销毁栈帧 */
    Leave();
    /* 退出中断 */
    InterruptReturn();
}
```

汇编调用C语言程序

系统调用
处理程序

其他函数的
嵌套调用

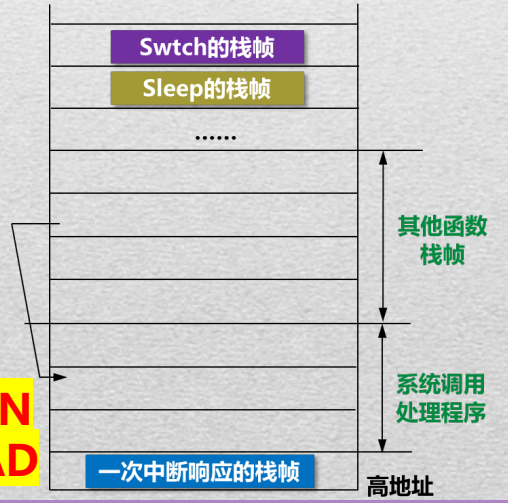
Process::Sleep

p_stat = SSLEEP/SWAIT
p_wchan = 睡眠原因

return

ProcessManager::Swtch

现场保护
恢复0#



p_stat = SRUN
p_flag有SLOAD

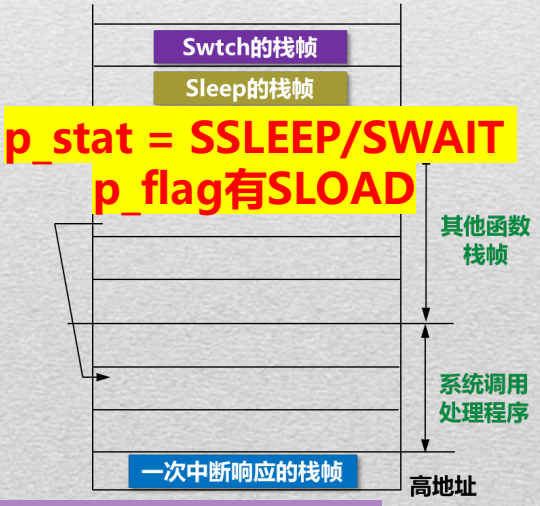
主要知识点：

现运行进程

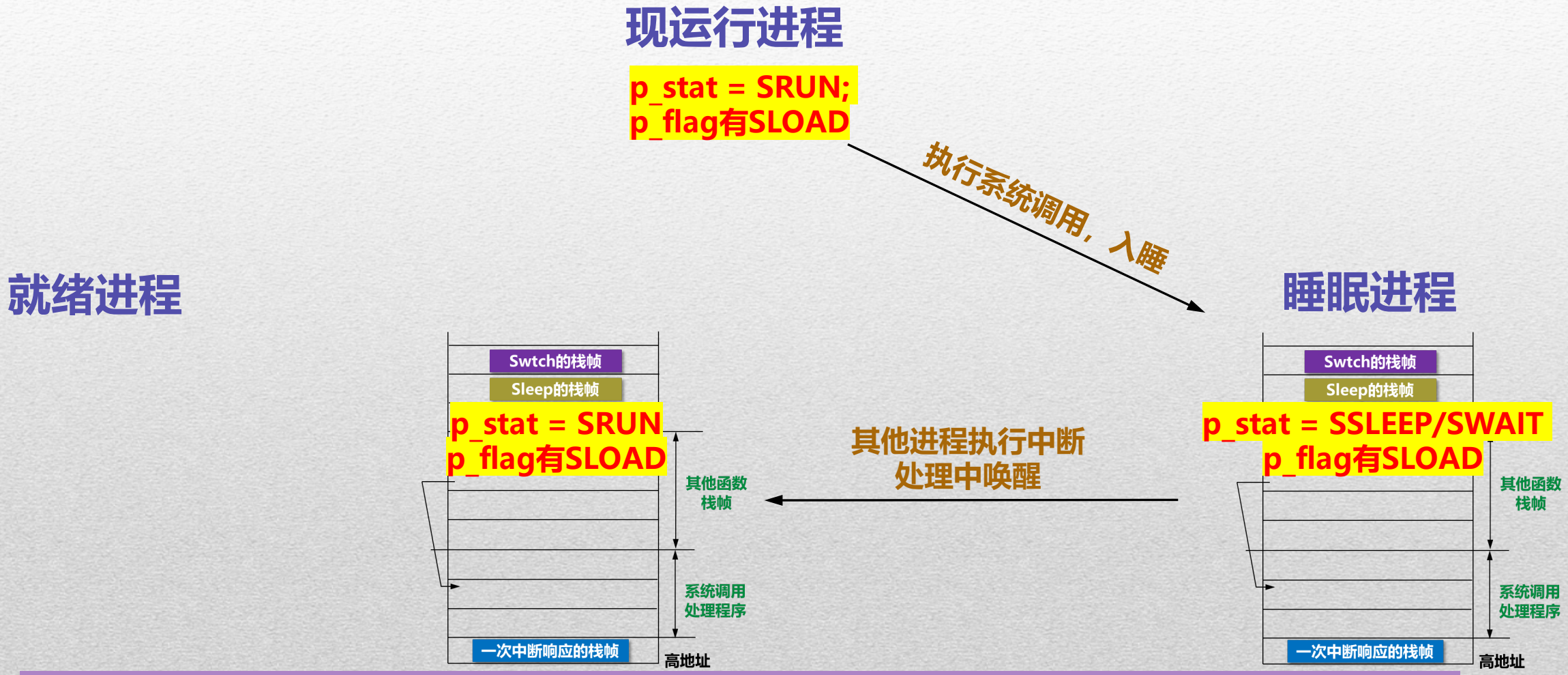
p_stat = SRUN;
p_flag有SLOAD

执行系统调用，入睡

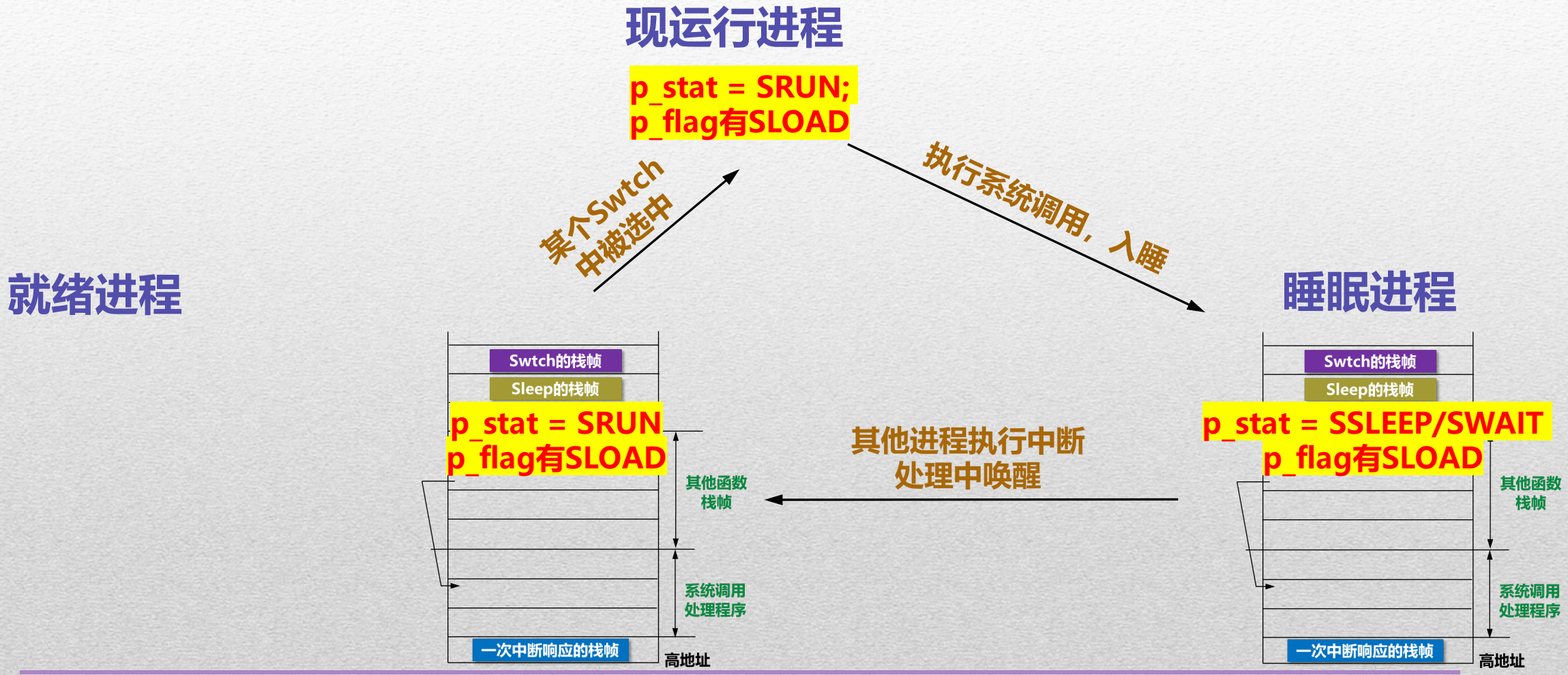
睡眠进程

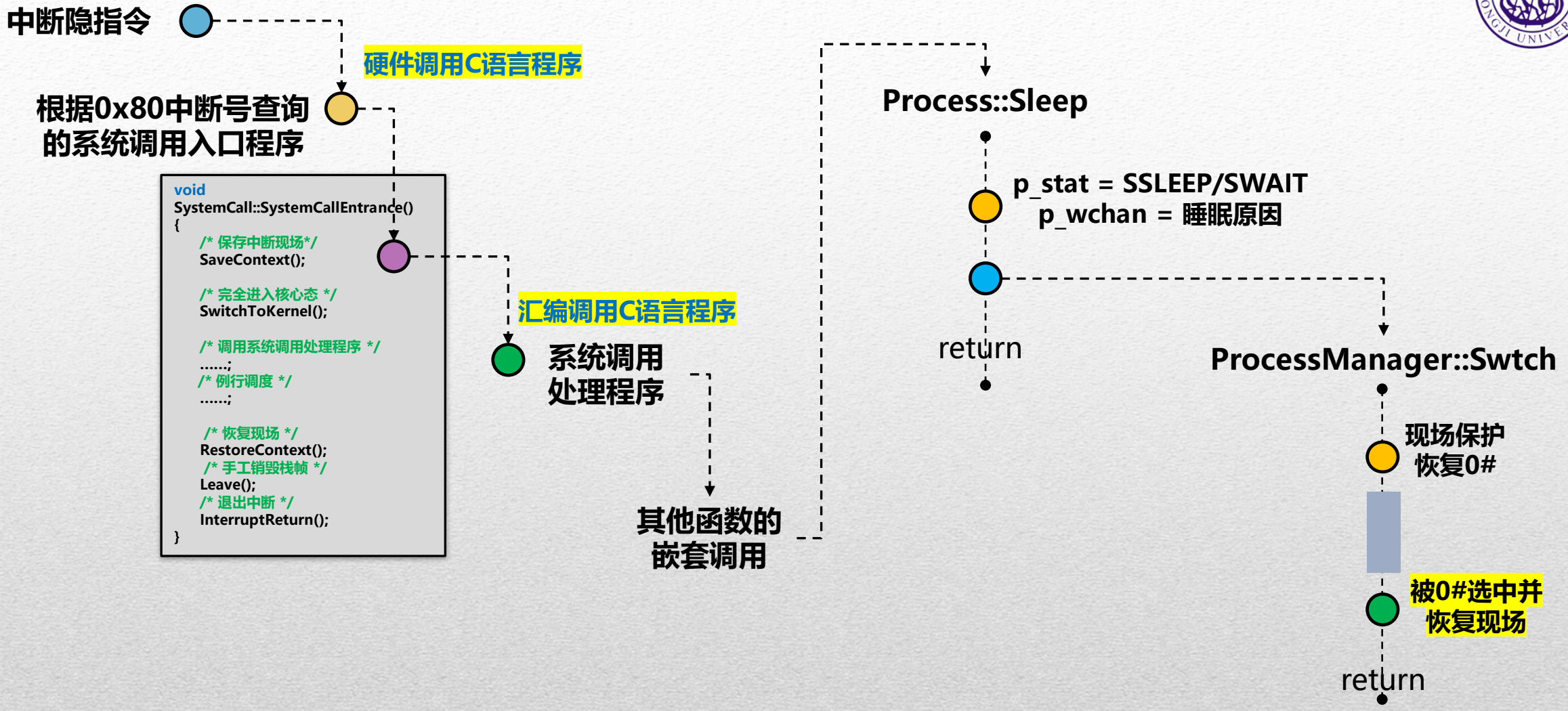


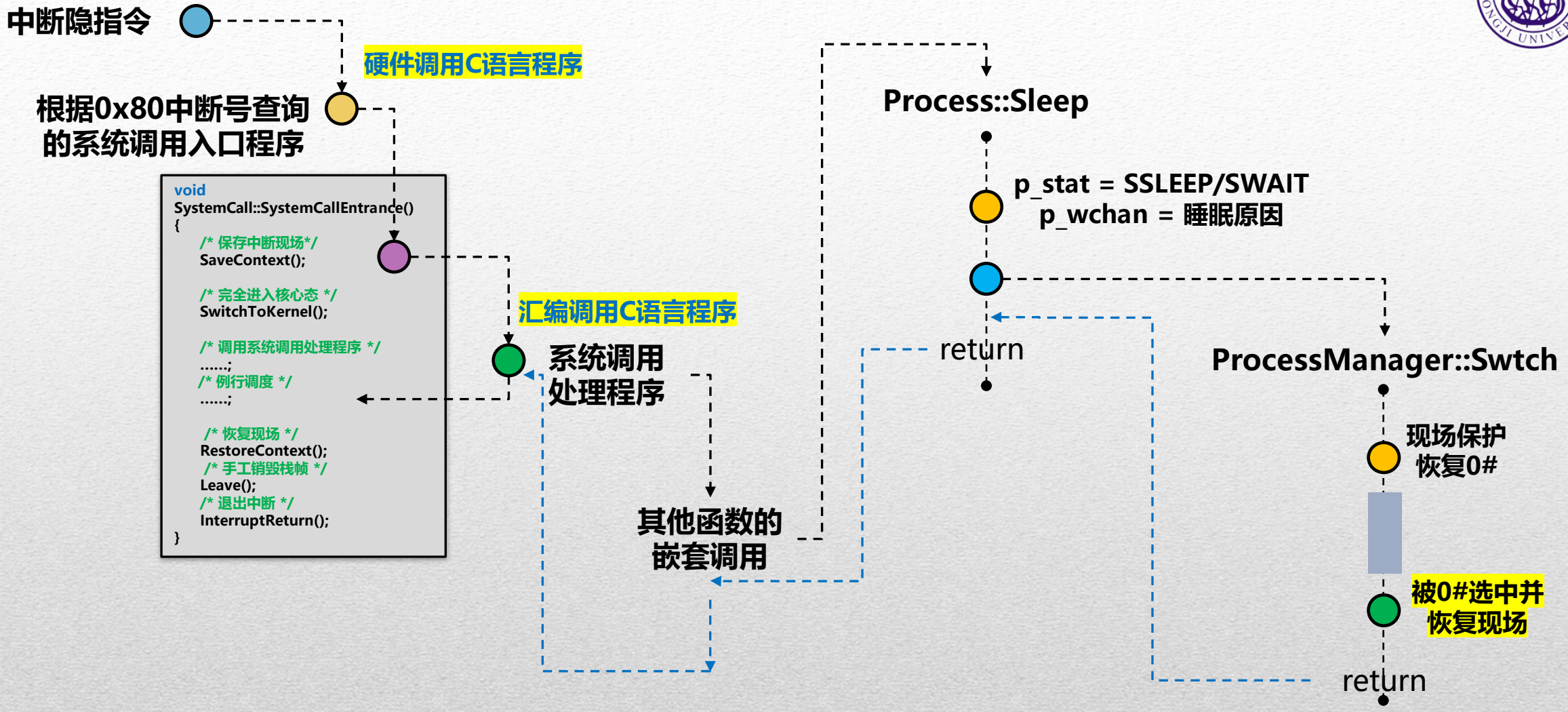
主要知识点：


现运行进程
p_stat = SRUN;
p_flag有SLOAD
执行系统调用，入睡
就绪进程
睡眠进程
Swtch的栈帧
Sleep的栈帧
p_stat = SRUN
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址
其他进程执行中断
处理中唤醒
Swtch的栈帧
Sleep的栈帧
p_stat = SSLEEP/SWAIT
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址

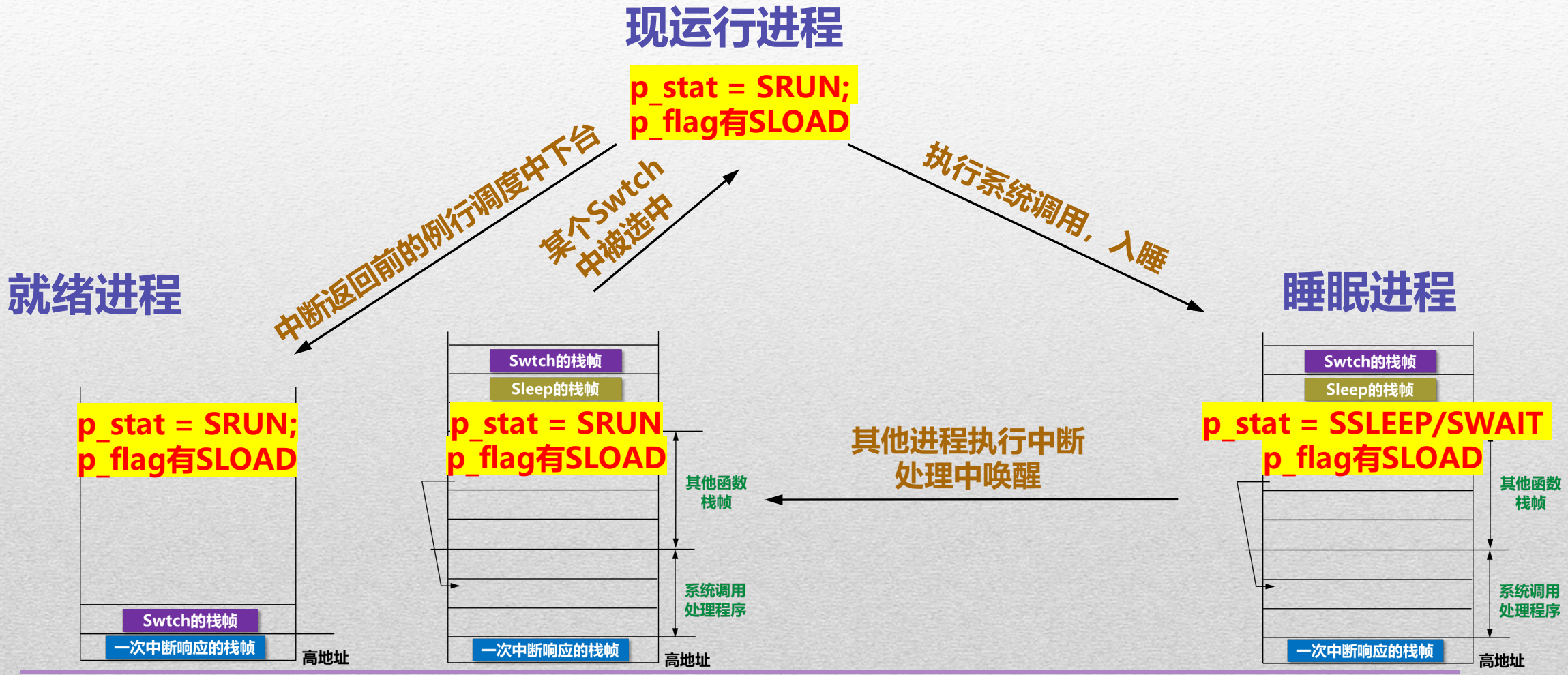
主要知识点：

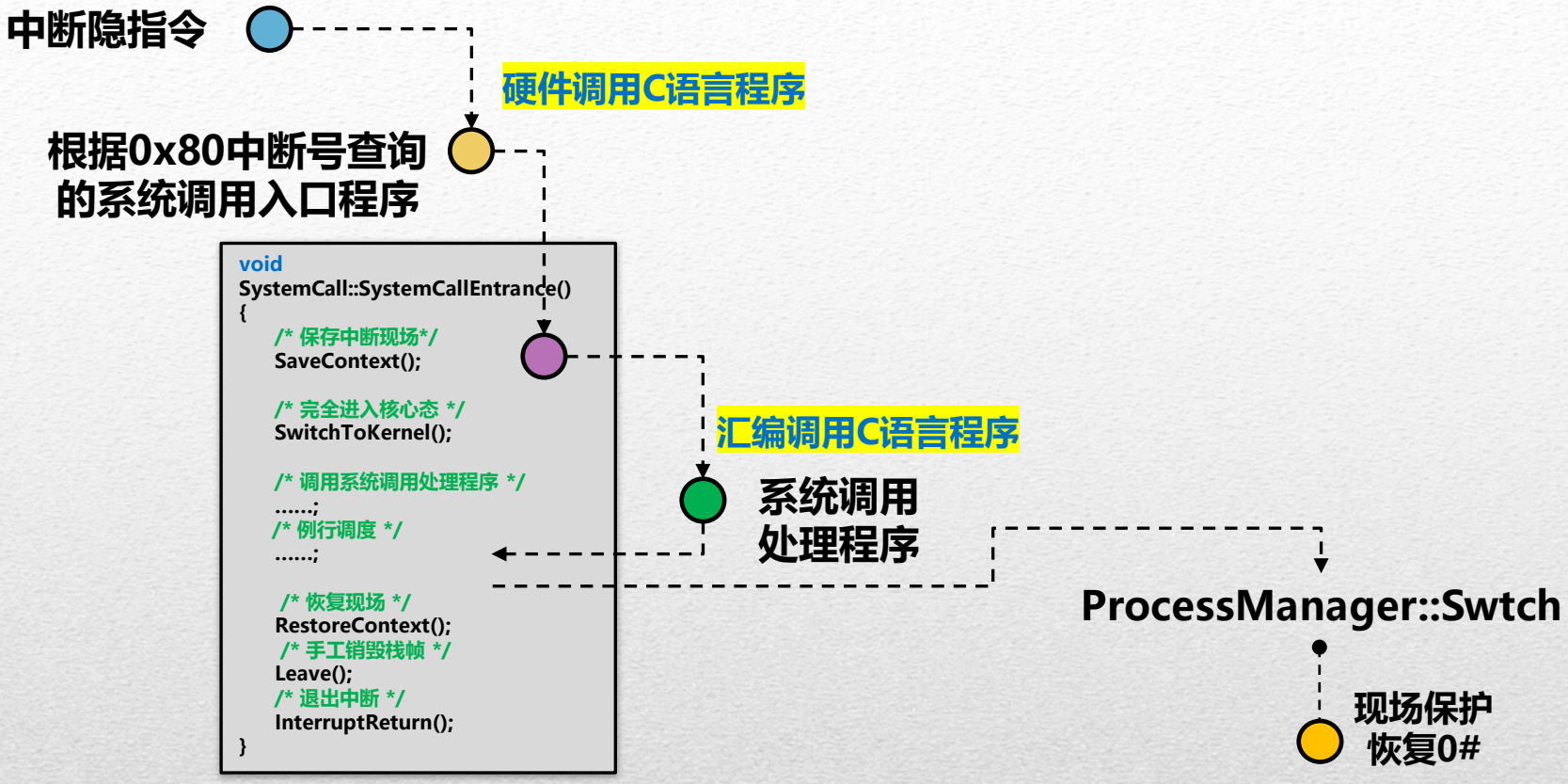

现运行进程
p_stat = SRUN;
p_flag有SLOAD
某个Swtch
中被选中
执行系统调用，入睡
就绪进程
睡眠进程
Swtch的栈帧
Sleep的栈帧
p_stat = SRUN
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址
其他进程执行中断
处理中唤醒
Swtch的栈帧
Sleep的栈帧
p_stat = SSLEEP/SWAIT
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址


中断隐指令
硬件调用C语言程序
根据0x80中断号查询
的系统调用入口程序
void
SystemCall::SystemCallEntrance()
{
/* 保存中断现场*/
SaveContext();
/* 完全进入核心态 */
SwitchToKernel();
/* 调用系统调用处理程序 */
……;
/* 例行调度 */
……;
/* 恢复现场 */
RestoreContext();
/* 手工销毁栈帧 */
Leave();
/* 退出中断 */
InterruptReturn();
}
汇编调用C语言程序
系统调用
处理程序
其他函数的
嵌套调用
Process::Sleep
p_stat = SSLEEP/SWAIT
p_wchan = 睡眠原因
return
ProcessManager::Swtch
现场保护
恢复0#
被0#选中并
恢复现场
return

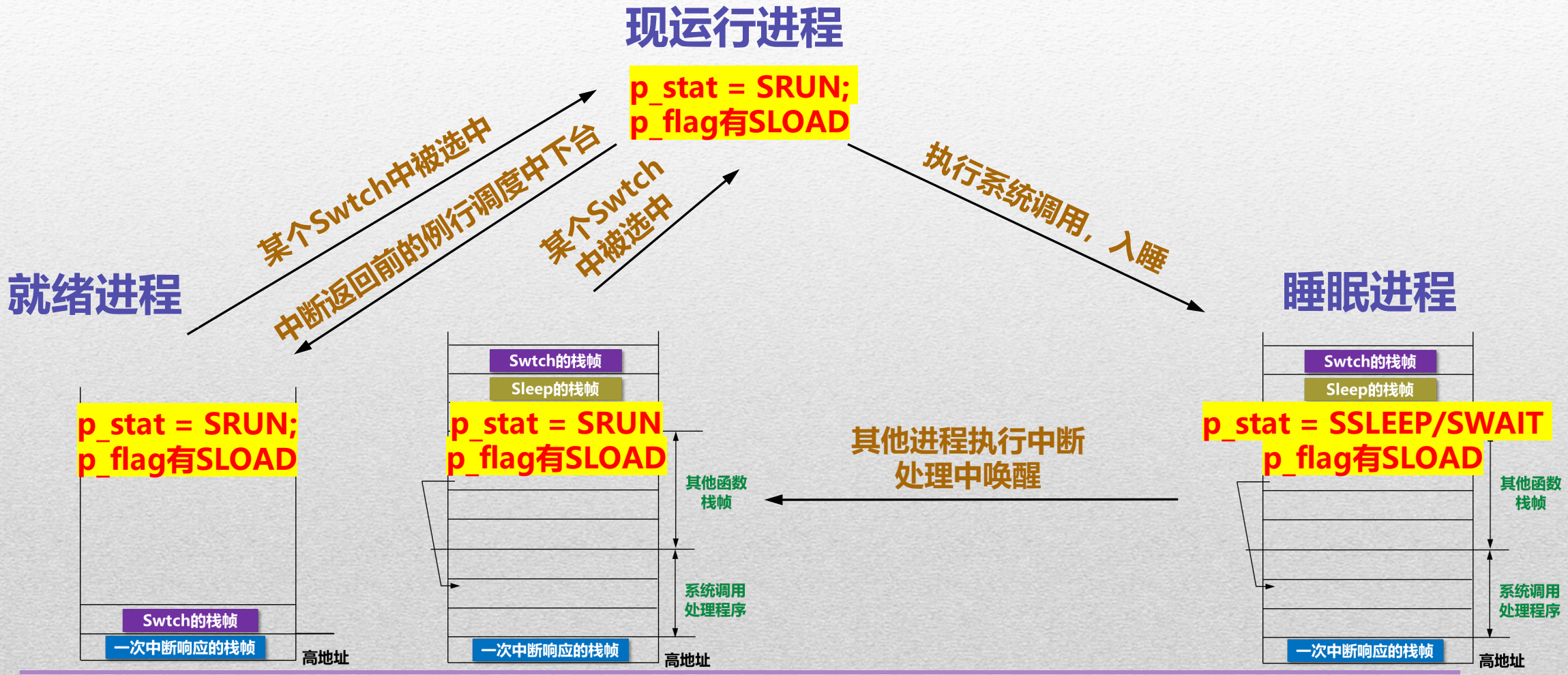

中断隐指令
硬件调用C语言程序
根据0x80中断号查询
的系统调用入口程序
void
SystemCall::SystemCallEntrance()
{
/* 保存中断现场*/
SaveContext();
/* 完全进入核心态 */
SwitchToKernel();
/* 调用系统调用处理程序 */
……;
/* 例行调度 */
……;
/* 恢复现场 */
RestoreContext();
/* 手工销毁栈帧 */
Leave();
/* 退出中断 */
InterruptReturn();
}
汇编调用C语言程序
系统调用
处理程序
其他函数的
嵌套调用
Process::Sleep
p_stat = SSLEEP/SWAIT
p_wchan = 睡眠原因
return
ProcessManager::Swtch
现场保护
恢复0#
被0#选中并
恢复现场
return

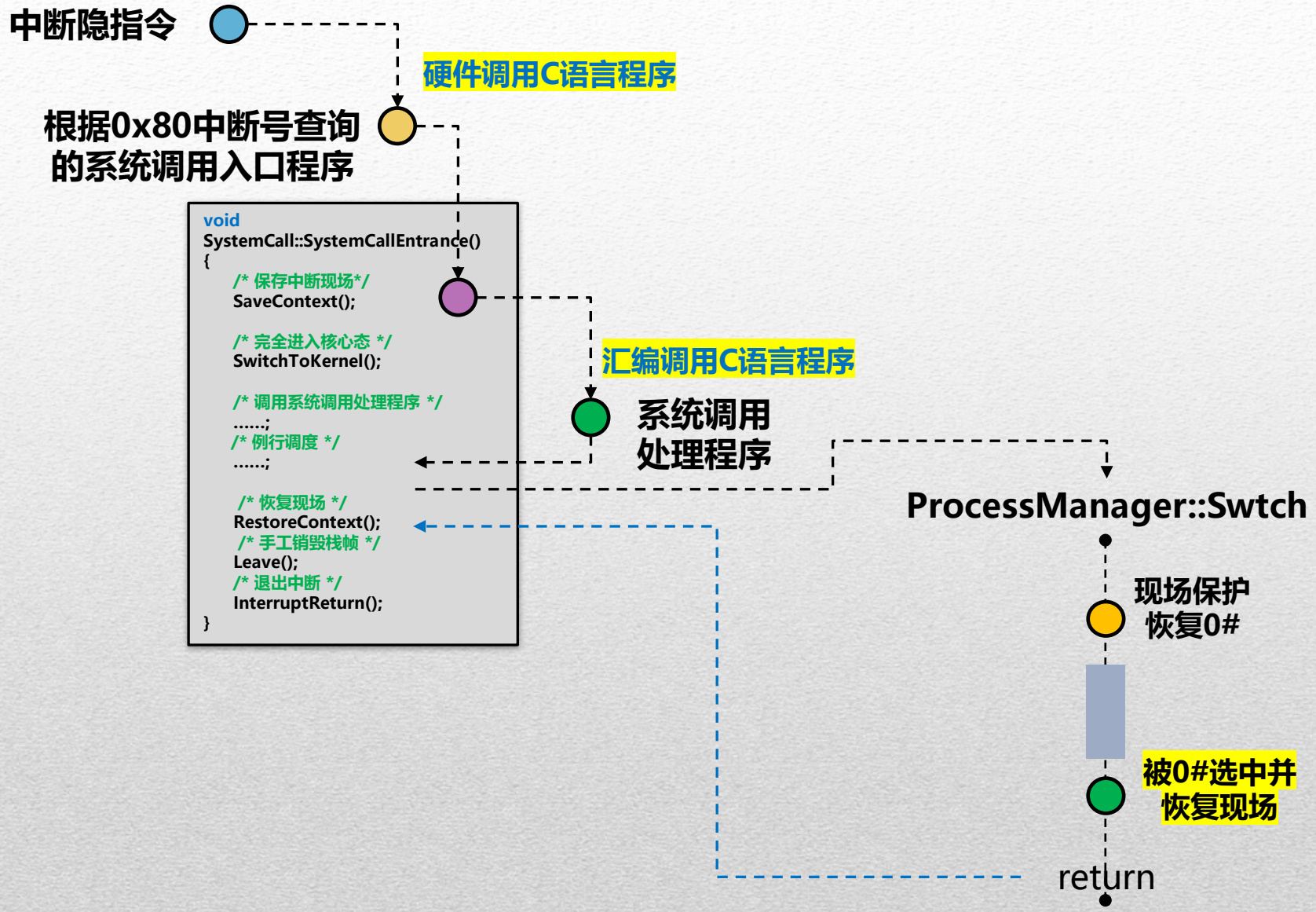
主要知识点：


现运行进程
p_stat = SRUN;
p_flag有SLOAD
中断返回前的例行调度中下台
某个Swtch
中被选中
执行系统调用，入睡
就绪进程
睡眠进程
p_stat = SRUN;
p_flag有SLOAD
Swtch的栈帧
一次中断响应的栈帧
高地址
Swtch的栈帧
Sleep的栈帧
p_stat = SRUN
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址
其他进程执行中断
处理中唤醒
Swtch的栈帧
Sleep的栈帧
p_stat = SSLEEP/SWAIT
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址


中断隐指令
硬件调用C语言程序
根据0x80中断号查询
的系统调用入口程序
void
SystemCall::SystemCallEntrance()
{
/* 保存中断现场*/
SaveContext();
/* 完全进入核心态 */
SwitchToKernel();
/* 调用系统调用处理程序 */
……;
/* 例行调度 */
……;
/* 恢复现场 */
RestoreContext();
/* 手工销毁栈帧 */
Leave();
/* 退出中断 */
InterruptReturn();
}
汇编调用C语言程序
系统调用
处理程序
ProcessManager::Swtch
现场保护
恢复0#

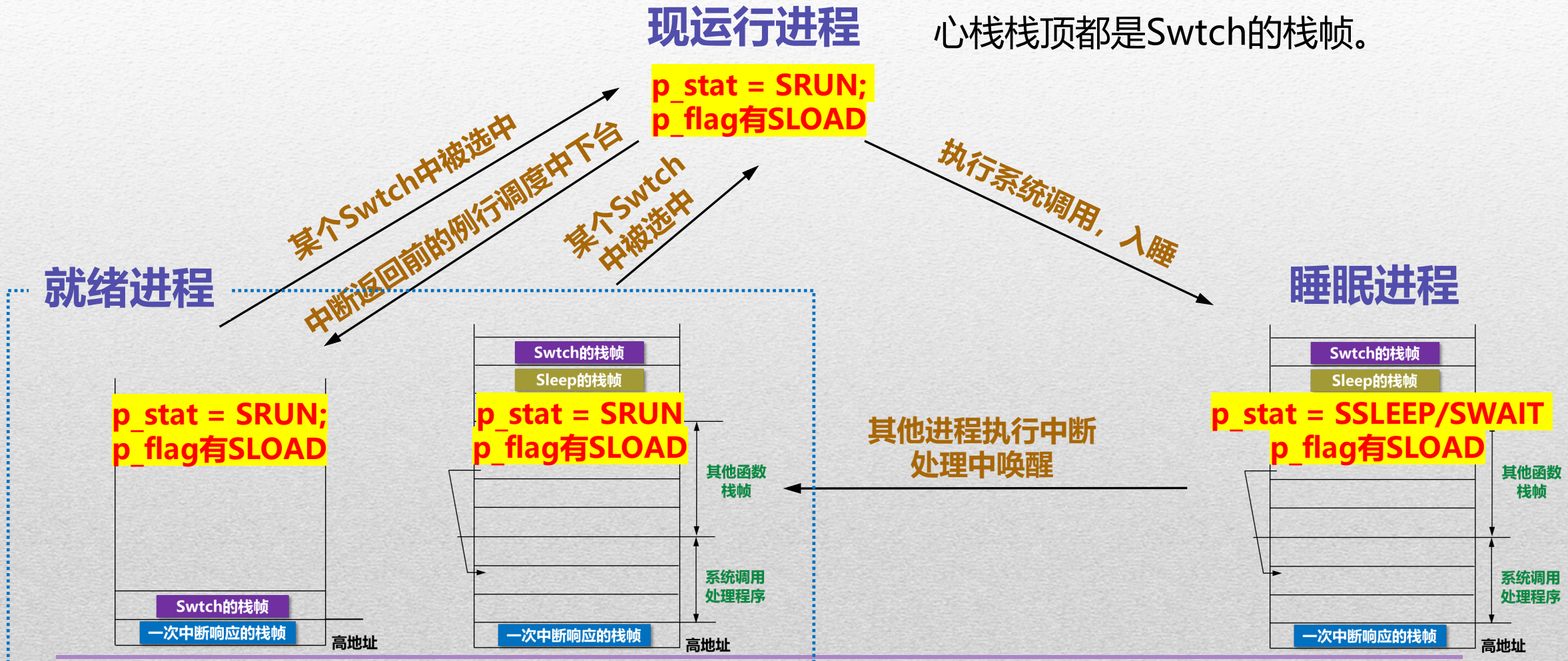
主要知识点：


现运行进程
p_stat = SRUN;
p_flag有SLOAD
某个Swtch中被选中
中断返回前的例行调度中下台
某个Swtch
中被选中
执行系统调用，入睡
就绪进程
睡眠进程
p_stat = SRUN;
p_flag有SLOAD
Swtch的栈帧
一次中断响应的栈帧
高地址
Swtch的栈帧
Sleep的栈帧
p_stat = SRUN
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址
其他进程执行中断
处理中唤醒
Swtch的栈帧
Sleep的栈帧
p_stat = SSLEEP/SWAIT
p_flag有SLOAD
其他函数
栈帧
系统调用
处理程序
一次中断响应的栈帧
高地址

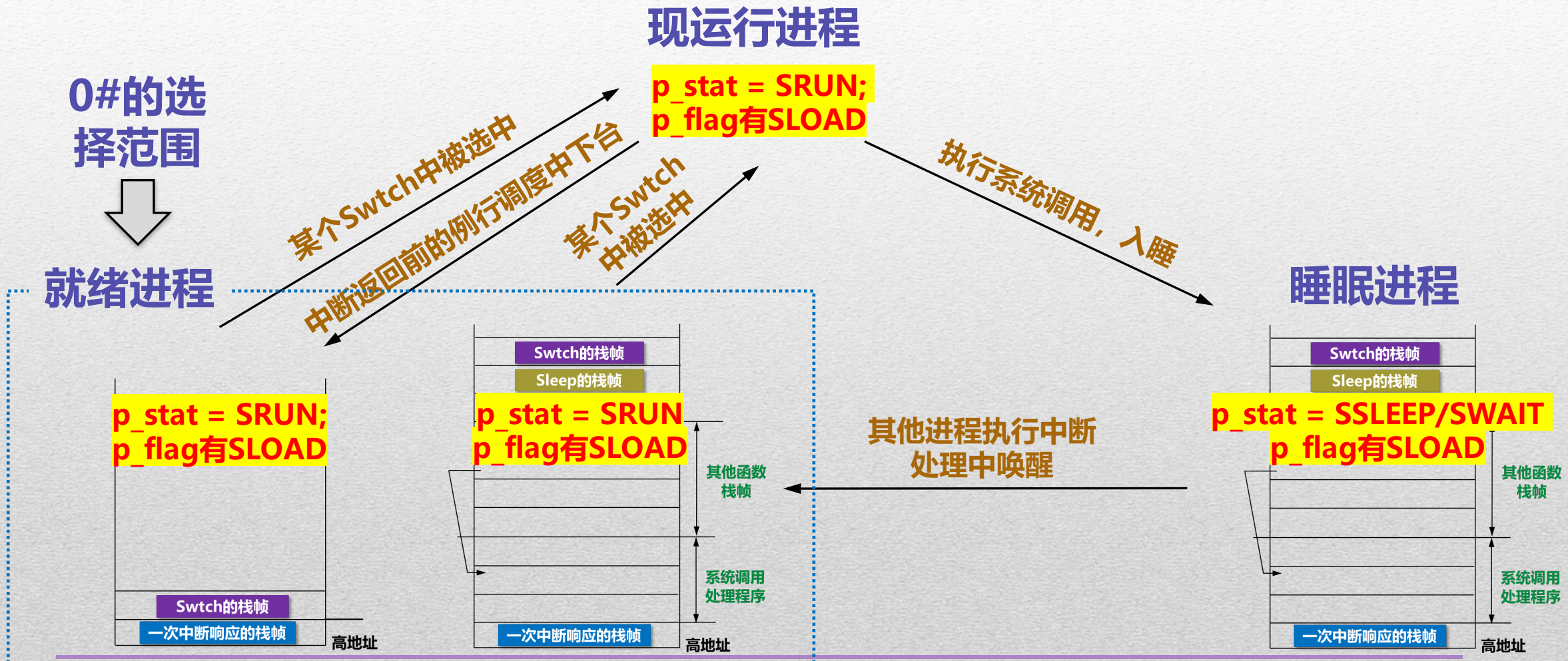

中断隐指令
硬件调用C语言程序
根据0x80中断号查询
的系统调用入口程序
void
SystemCall::SystemCallEntrance()
{
/* 保存中断现场*/
SaveContext();
/* 完全进入核心态 */
SwitchToKernel();
/* 调用系统调用处理程序 */
……;
/* 例行调度 */
……;
/* 恢复现场 */
RestoreContext();
/* 手工销毁栈帧 */
Leave();
/* 退出中断 */
InterruptReturn();
}
汇编调用C语言程序
系统调用
处理程序
ProcessManager::Swtch
现场保护
恢复0#
被0#选中并
恢复现场
return

主要知识点：

所有的调度状态改变都发生在核心态，所以除现运行进程和新进程外，其余状态（就绪或睡眠）的进程核心栈栈顶都是Swtch的栈帧。



主要知识点：


现运行进程
p_stat = SRUN;
p_flag有SLOAD
0#的选择范围
就绪进程
某个Swtch中被选中
中断返回前的例行调度中下台
某个Swtch中被选中
执行系统调用，入睡
睡眠进程
其他进程执行中断处理中唤醒
p_stat = SRUN;
p_flag有SLOAD
Swtch的栈帧
一次中断响应的栈帧
高地址
Swtch的栈帧
Sleep的栈帧
p_stat = SRUN
p_flag有SLOAD
其他函数栈帧
系统调用处理程序
一次中断响应的栈帧
高地址
Swtch的栈帧
Sleep的栈帧
p_stat = SSLEEP/SWAIT
p_flag有SLOAD
其他函数栈帧
系统调用处理程序
一次中断响应的栈帧
高地址

（1）下列关于外部I/O中断的叙述中，正确的是（　　）

A. 中断控制器按所接收中断请求的先后次序进行中断优先级排队

B. CPU响应中断时，通过执行中断隐指令完成通用寄存器的保护

C. CPU只有在处于中断允许状态时，才能响应外部设备的中断请求

D. 有中断请求时，CPU立即暂停当前指令执行，转去执行中断服务程序

（2）下列关于多重中断系统的叙述中，错误的是（　　）

A. 在一条指令执行结束时响应中断

B. 中断处理期间CPU一直处于关中断状态

C. 外设中断请求的产生与当前指令的执行无关

D. CPU通过采样中断请求信号检测中断请求

（3）现运行进程在核心态运行时不响应中断吗?

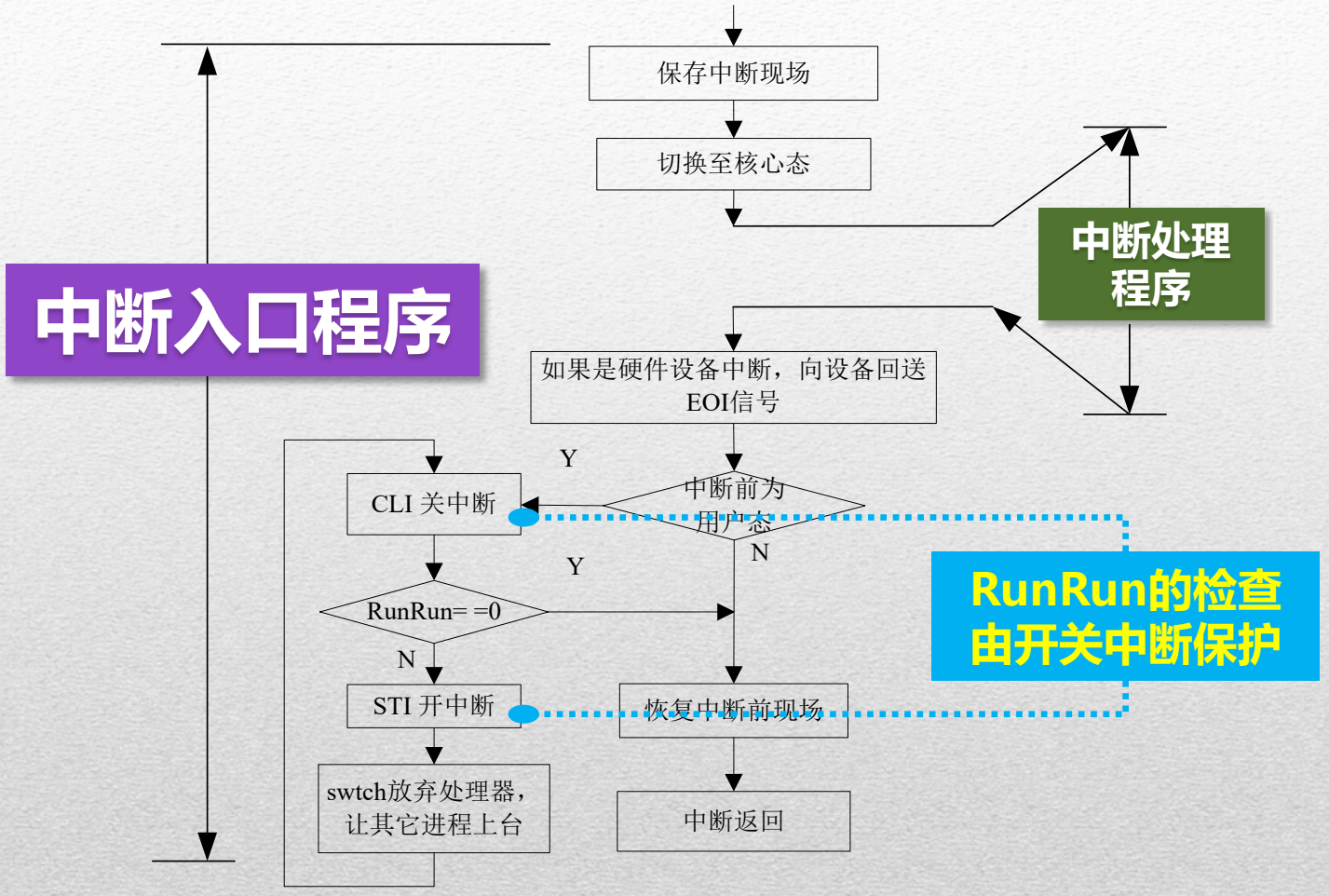
不是的。CPU只要开中断，就会响应从8259转发的中断请求。

核心态响应中断与用户态响应中断的区别？？？

（4）CPU执行应用程序时，处理机优先级是多少？系统响应键盘中断吗?

CPU执行应用程序时，处理机优先级为0。

只要CPU在开中断的状态，就可以响应任何中断。



1. 既然核心态下 不会发生进程切换，为什么检测runurn要类似临界处理?

2. 如果RunRun不为0，中断什么时候开?