



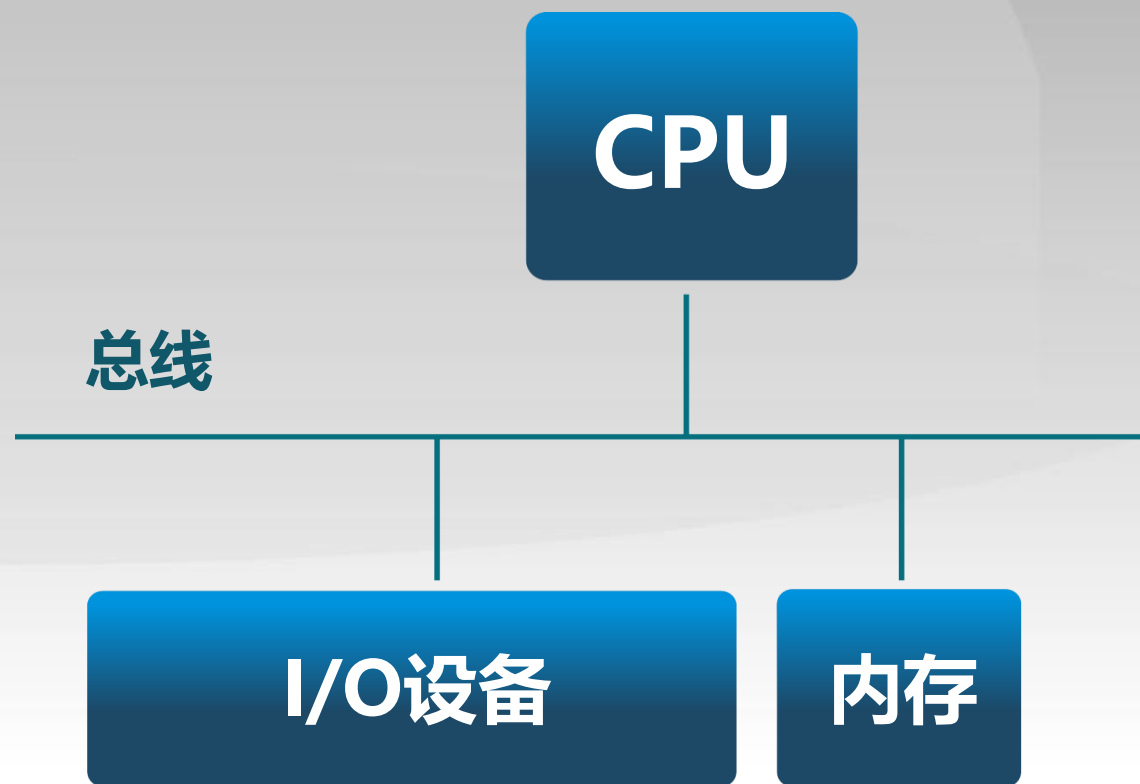
操作系统

Operating System

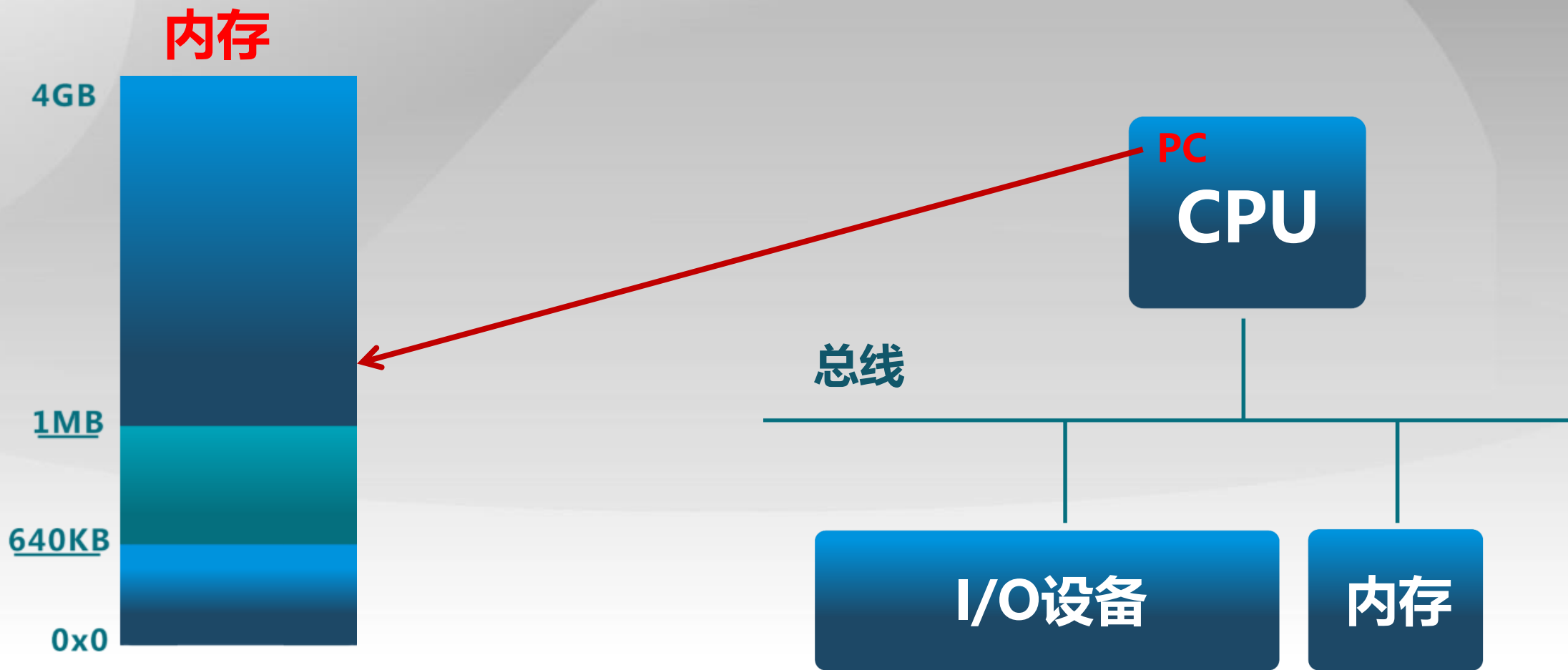
内容

- ▣ 启动前的准备
- ▣ 启动流程
- ▣ 中断/异常/系统调用
- ▣ 系统调用示例
- ▣ ucore+系统调用代码

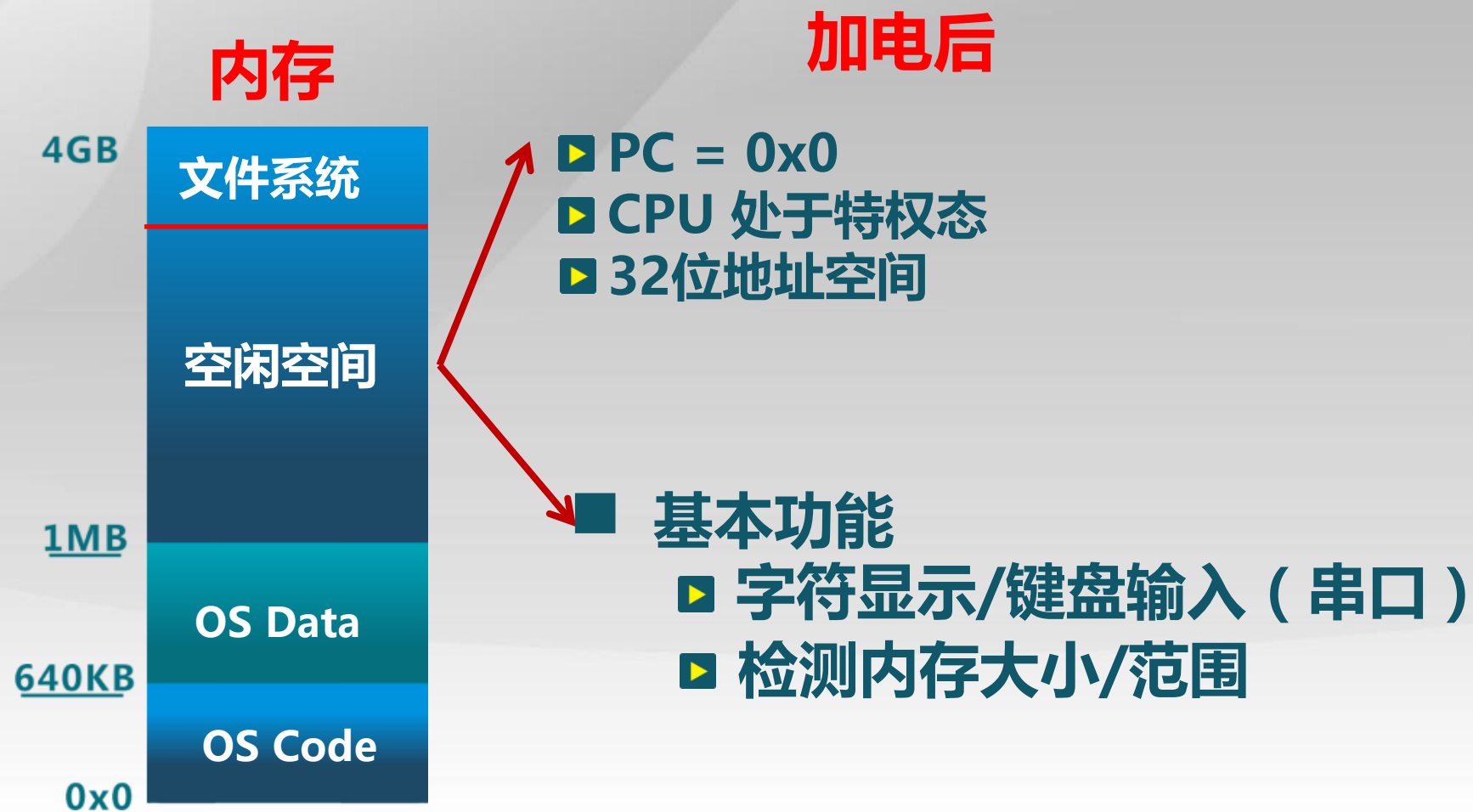
计算机体系结构概述



计算机体系结构概述



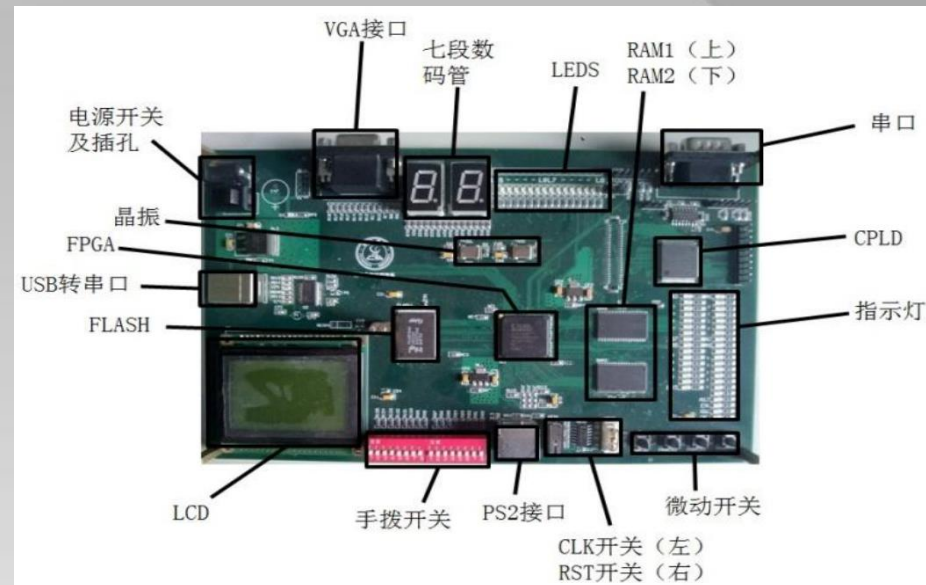
启动前的准备



启动前的准备

系统加电启动后，MIPS处理器默认的程序入口是0xBFC00000(虚拟地址)，此地址在KSEG1(无缓存)区域内，对应的物理地址是0x1FC00000(高3位清零)，所以CPU从物理地址0x1FC00000开始取第一条指令，这个地址在硬件上已经确定为FLASH(BIOS)的位置

- CPU状态寄存器初始化
- MMU/TLBs初始化成无效值
- 其它寄存器赋初始值
- CPU协处理器CP0寄存器初始化
- 硬件初始化的最主要工作是内存控制器的初始化
- 加载程序/OS并跳转到程序/OS入口处



启动前的准备

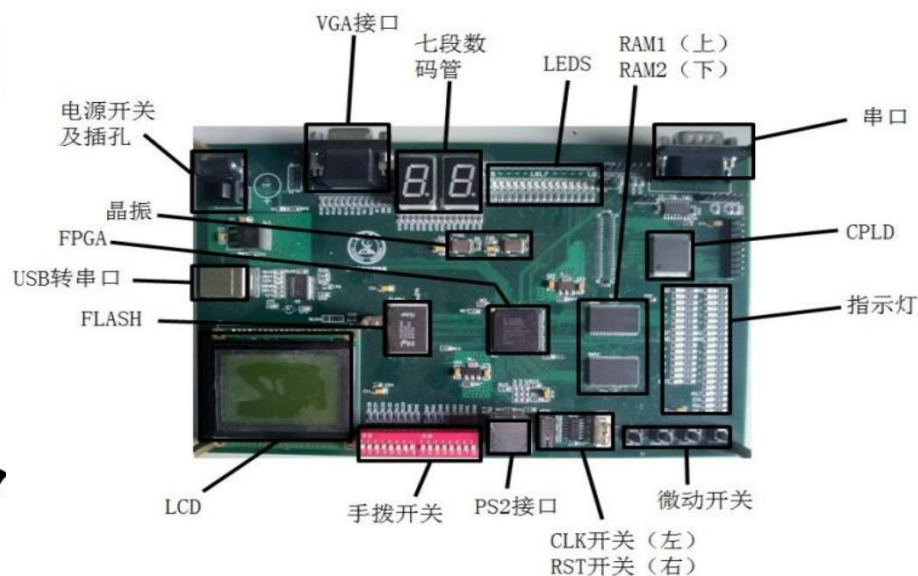
程序如何装入？

❊ 程序装入到FLASH中

- ❑ 采用提供的软件装入
- ❑ CPU首先将其boot到RAM1中
- ❑ 再从RAM1中运行

❊ 程序直接装入到RAM1中

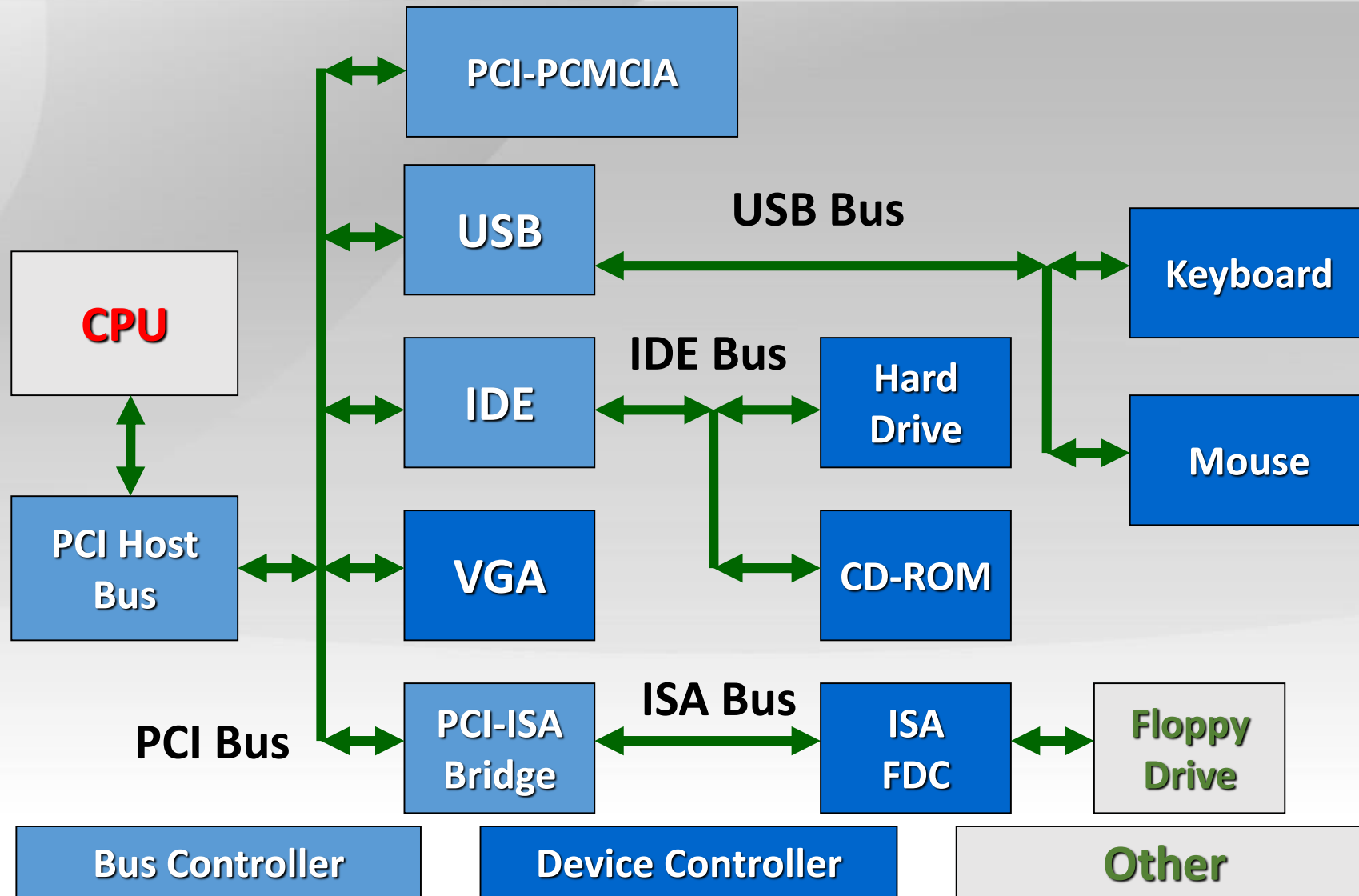
- ❑ 直接用软件装入到RAM1中



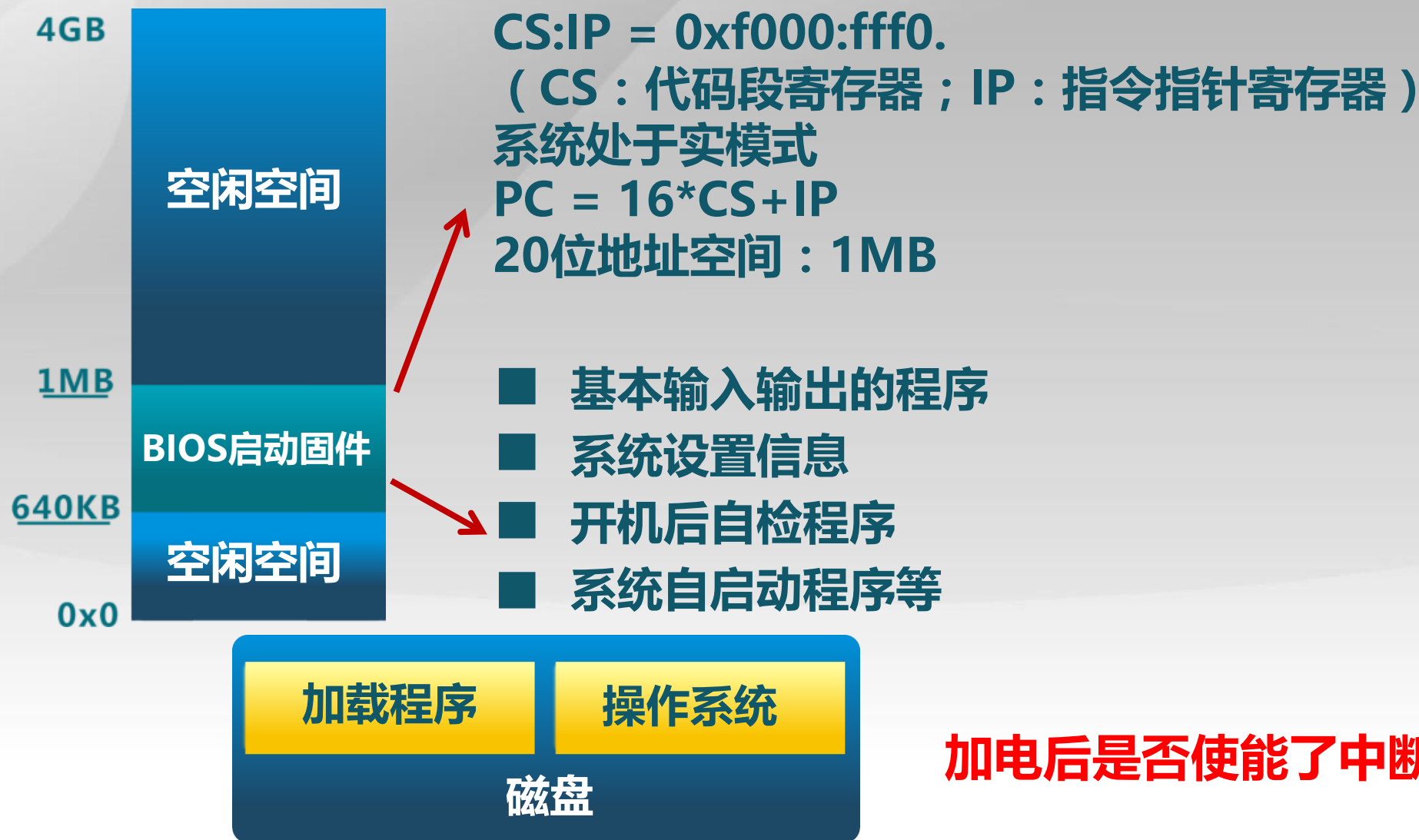
启动前的准备

功能区	地址段	说明
系统程序区	0x0000~0x3FFF	监控程序
用户程序区	0x4000~0x7FFF (32K)	存放用户程序
系统数据区	0x8000~0xBEFF	监控程序使用的数据区
Com1数据端口/命令端口	0xBF00~0xBF01	
Com2数据端口/命令端口	0xBF02~0xBF03	第2个串口的端口
预留给其他接口	0xBF04~0xBF0F	保留
系统堆栈区	0xBF10~0xBFFF	用于系统堆栈
用户数据区	0xC000~0xFFFF	用户程序使用的数据区

X86 Typical System

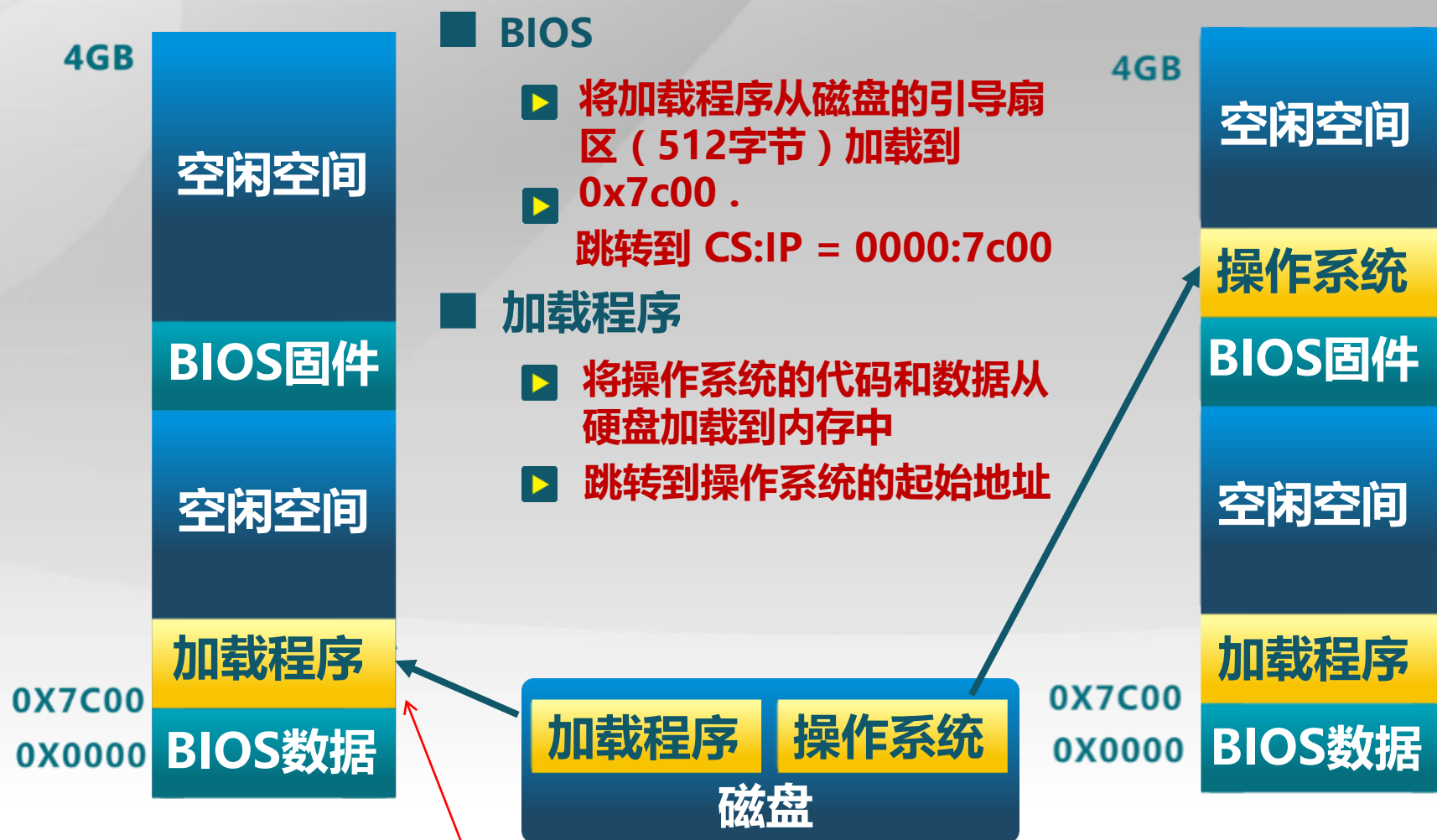


启动时计算机内存和磁盘布局



加电后是否使能了中断机制？理由？

加载程序的内存地址空间



可能有不同的文件系统，BIOS需要支持不同的文件系统

BIOS系统调用

- BIOS以中断调用的方式 提供了基本的I/O功能
 - ▣ INT 10h: 字符显示
 - ▣ INT 13h: 磁盘扇区读写
 - ▣ INT 15h: 检测内存大小
 - ▣ INT 16h: 键盘输入
- 只能在x86的实模式下访问

总结：加电后读BIOS BIOS读加载程序 加载程序读内核映像



操作系统

Operating System

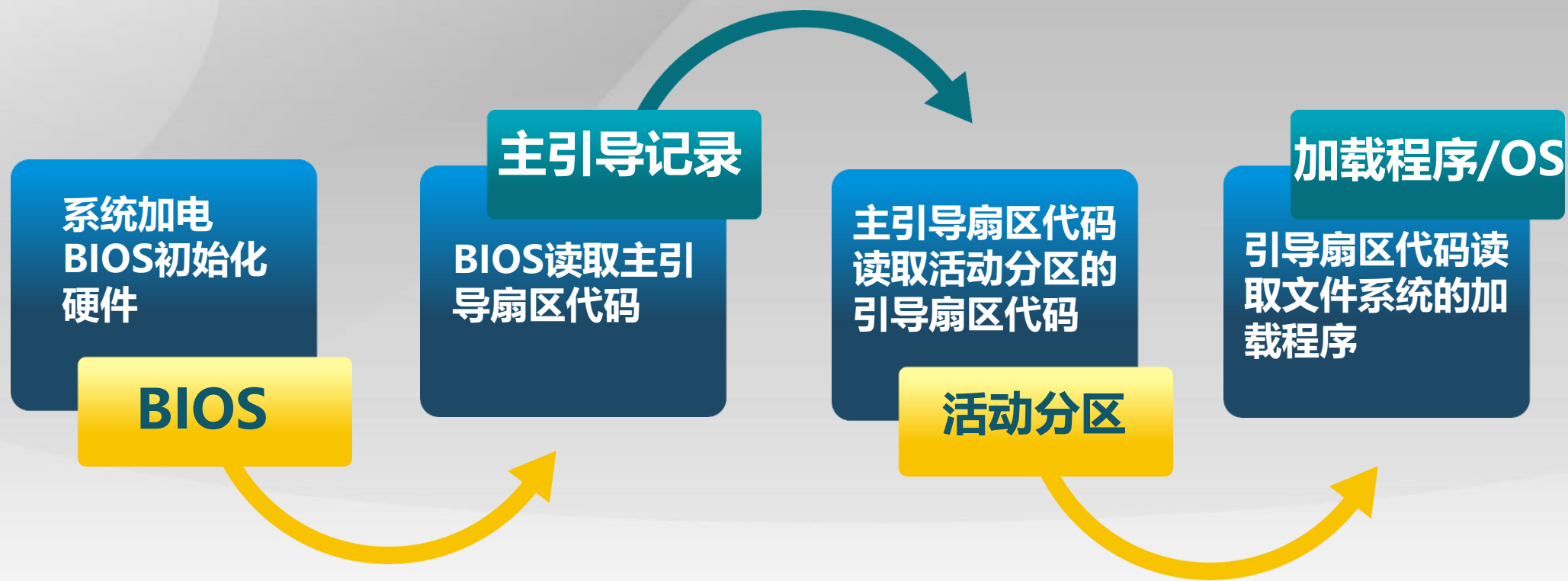
计算机启动流程

一般计算机启动过程描述



计算机启动流程

x86一般启动过程描述



CPU初始化

← 加电后读BIOS的细化

- CPU加电稳定后从0xFFFF0读第一条指令
 - ▣ CS:IP = 0xf000:fff0
 - ▣ 第一条指令是跳转指令
- CPU初始状态为16位实模式
 - ▣ CS:IP是16位寄存器
 - ▣ 指令指针PC = 16*CS+IP
 - ▣ 最大地址空间是1MB

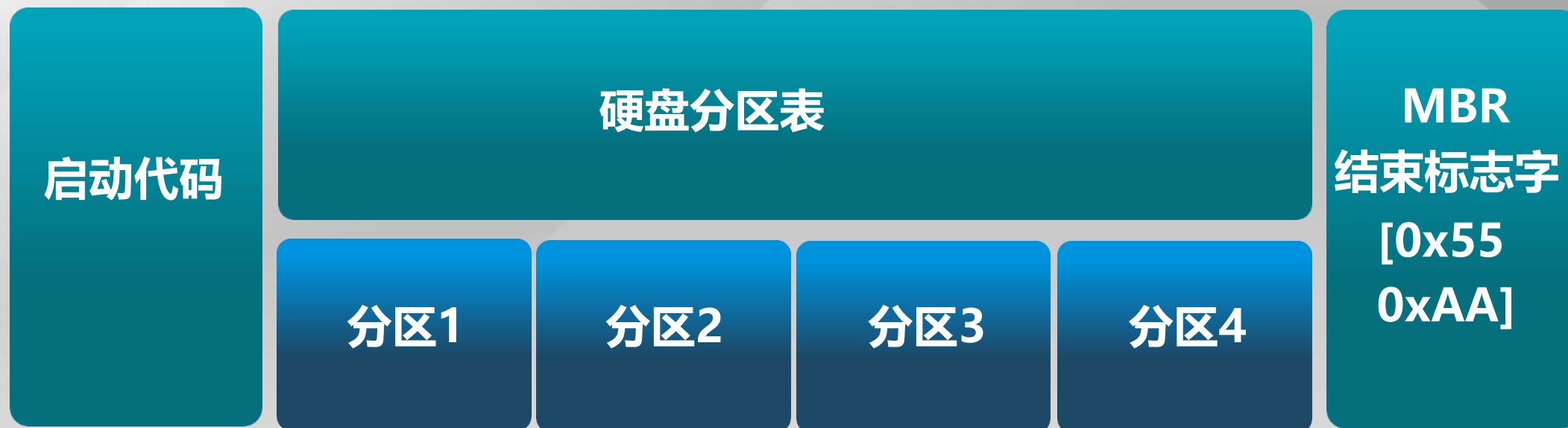
BIOS初始化过程

BIOS读加载程序的细化

- 硬件自检POST
- 检测系统中内存和显卡等关键部件的存在和工作状态
- 查找并执行显卡等接口卡BIOS，进行设备初始化；
- 执行系统BIOS，进行系统检测；
 - ▣ 检测和配置系统中安装的即插即用设备；
- 更新CMOS中的扩展系统配置数据ESCD
- 按指定启动顺序从软盘、硬盘或光驱启动

ESCD每次加电都要更新

主引导记录MBR格式



■ 启动代码：446字节

- ▣ 检查分区表正确性
- ▣ 加载并跳转到磁盘上的引导程序

■ 硬盘分区表：64字节

- ▣ 描述分区状态和位置
- ▣ 每个分区描述信息占据16字节

■ 结束标志字：2字节(55AA)

- ▣ 主引导记录的有效标志

分区引导扇区格式

JMP

文件卷
头结构

启动代码

结束标志
[0x55
0xAA]

- 跳转指令：跳转到启动代码
 - ▣ 与平台相关代码
- 文件卷头：文件系统描述信息
- 启动代码：跳转到加载程序
- 结束标志：55AA

加载程序(bootloader)

加载程序读内核映像的细化



系统启动规范

■ BIOS

- ▣ 固化到计算机主板上的程序
- ▣ 包括系统设置、自检程序和系统自启动程序
- ▣ BIOS-MBR、BIOS-GPT、PXE

BIOS单分区 多分区 网络协议栈

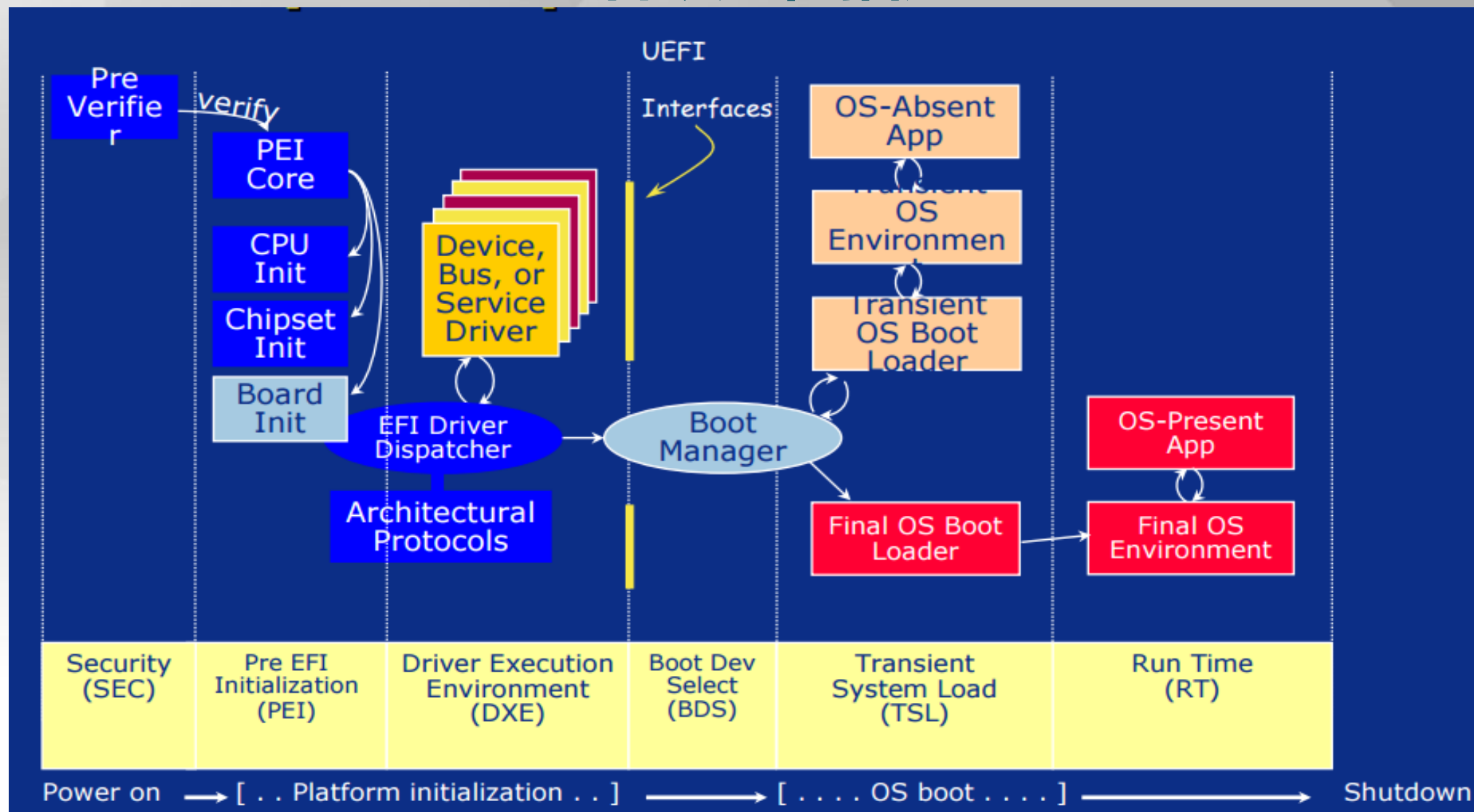
■ UEFI

- ▣ 接口标准
- ▣ 在所有平台上一致的操作系统启动服务

UEFI可信性检查：加入签名，只读入可信的引导记录

计算机启动流程

UEFI 启动过程描述





操作系统

Operating System

背景

- 为什么需要中断、异常和系统调用
 - ▣ 在计算机运行中，内核是被信任的第三方
 - ▣ 只有内核可以执行特权指令
 - ▣ 方便应用程序
- 中断和异常希望解决的问题
 - ▣ 当外设连接计算机时，会出现什么现象？
 - ▣ 当应用程序处理意想不到的行为时，会出现什么现象？
- 系统调用希望解决的问题
 - ▣ 用户应用程序是如何得到系统服务？
 - ▣ 系统调用和功能调用的不同之处是什么？

中断、异常和系统调用比较

■ 源头

- ▶ 中断：外设
- ▶ 异常：应用程序意想不到的行为
- ▶ 系统调用：应用程序请求操作提供服务

■ 响应方式

- ▶ 中断：异步
- ▶ 异常：同步
- ▶ 系统调用：异步或同步

■ 处理机制

- ▶ 中断：持续，对用户应用程序是透明的
- ▶ 异常：杀死或者重新执行意想不到的应用程序指令
- ▶ 系统调用：等待和持续

背景

- 基本的中断、异常和系统调用（10个）
 - ▣ FMEM, // bad physical address
 - ▣ FTIMER, // timer interrupt
 - ▣ FKEYBD, // keyboard interrupt
 - ▣ FPRIV, // privileged instruction
 - ▣ FINST, // illegal instruction
 - ▣ FSYS, // software trap
 - ▣ FARITH, // arithmetic trap
 - ▣ FIPAGE, // page fault on opcode fetch
 - ▣ FWPAGE, // page fault on write
 - ▣ FRPAGE, // page fault on read

背景

■ MIPS的中断、异常和系统调用

- 0: Interrupt, 中断;
- 1: TLB Modified, 试图修改TLB中映射为只读的内存地址;
- 2: TLB Miss Load, 试图读取一个没有在TLB中映射到物理地址的虚拟地址;
- 3: TLB Miss Store, 试图向一个没有在TLB中映射到物理地址的虚拟地址存入数据;
- 4: Address Error Load, 试图从一个非对齐的地址读取信息;
- 5: Address Error Store, 试图向一个非对齐的地址写入信息;
- 6: Instruction Bus Error, 一般是指令Cache出错;
- 7: Data Bus Error, 一般是数据Cache出错;
- 8: Syscall, 由syscall指令产生。操作系统下, 通用的由用户态进入内核态的方法。
- 9: Break Point, 由break指令产生。
- 23: Watch, 内存断点异常。
-

背景

■ x86的中断、异常和系统调用（80386最多处理256种中断或异常）

- 80386有两根引脚INTR和NMI接受外部中断请求信号。
- INTR接受可屏蔽中断请求。NMI接受不可屏蔽中断请求。
- 在80386中，标志寄存器EFLAGS中的IF标志决定是否屏蔽可屏蔽中断请求。
- 异常进一步分类为故障(Fault)、陷阱(Trap)和中止(Abort)

80386响应 中断/异常 的优先级	中断/异常类型	优先级
	调试故障	最高
	其它故障	↓
	陷阱指令INT n和INTO	↓
	调试陷阱	↓
	NMI中断	↓
	INTR中断	最低

背景

■ x86的中断、异常和系统调用（80386最多处理256种中断或异常）

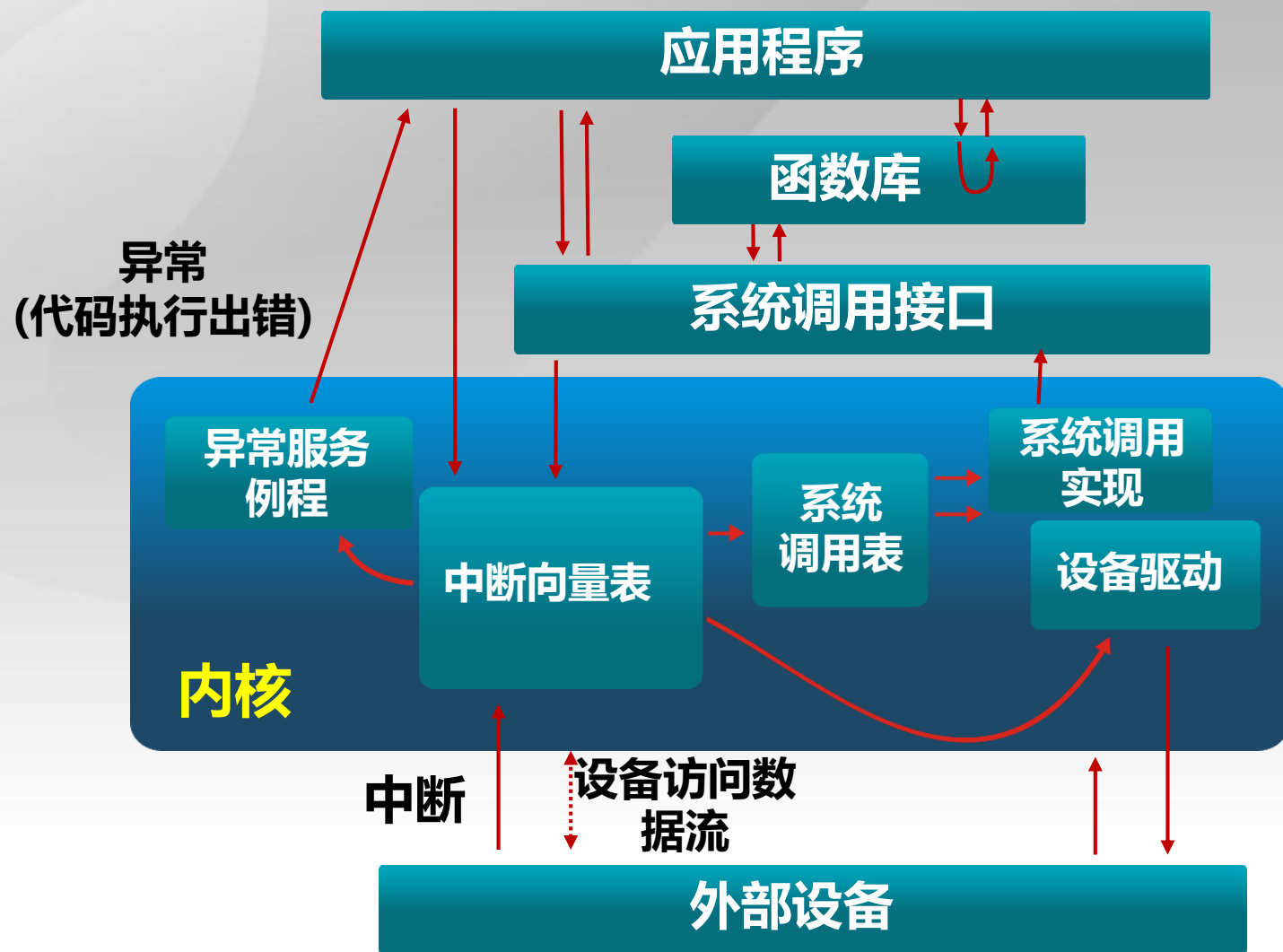
向量号	异常名称	异常类型	出错代码	相关指令
0	除法出错	故障	无	DIV, IDIV
1	调试异常	故障/陷阱	无	任何指令
3	单字节INT3	陷阱	无	INT 3
4	溢出	陷阱	无	INTO
5	边界检查	故障	无	BOUT
6	非法操作码	故障	无	非法指令编码或操作数
7	设备不可用	故障	无	浮点指令或WAIT
8	双重故障	中止	有	任何指令
9	协处理器段越界	中止	无	访问存储器的浮点指令
0AH	无效TSS异常	故障	有	JMP、CALL、IRET或中断
0BH	段不存在	故障	有	装载段寄存器的指令

背景

■ x86的中断、异常和系统调用（80386最多处理256种中断或异常）

向量号	异常名称	异常类型	出错代码	相关指令
0CH	堆栈段异常	故障	有	装载SS寄存器的任何指令、对SS寻址的段访问的任何指令
0DH	通用保护异常	故障	有	任何特权指令、任何访问存储器的指令
0EH	页异常	故障	有	任何访问存储器的指令
10H	协处理器出错	故障	无	浮点指令或WAIT
11H—0FFH	软中断	陷阱	无	INT n

内核的进入与退出



中断、异常和系统调用比较

■ 源头

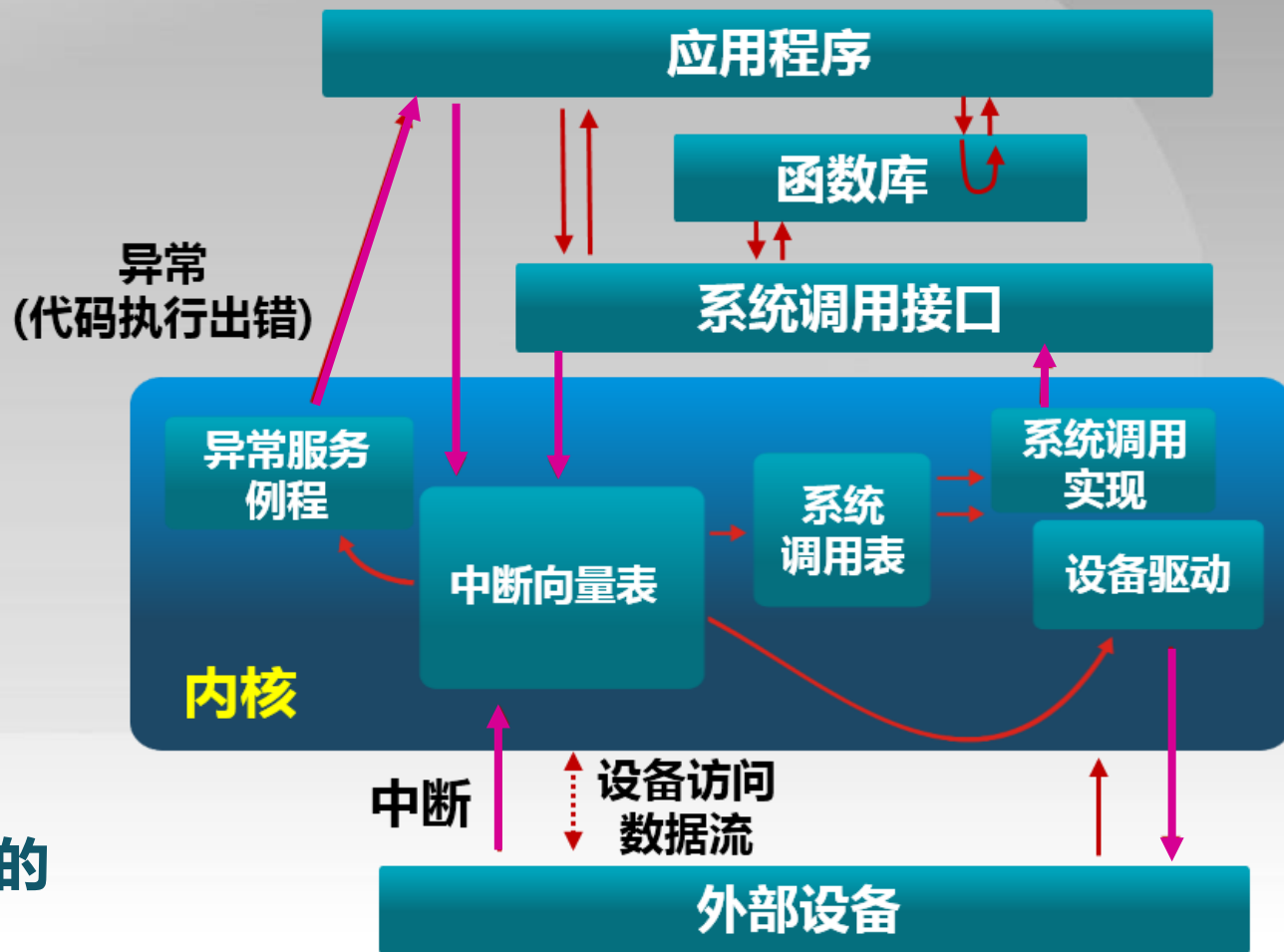
- ▶ 中断：外设
- ▶ 异常：应用程序意想不到的行为
- ▶ 系统调用：应用程序请求操作提供服务

■ 响应方式

- ▶ 中断：异步
- ▶ 异常：同步
- ▶ 系统调用：异步或同步

■ 处理机制

- ▶ 中断：持续，对用户应用程序是透明的
- ▶ 异常：杀死或者重新执行意想不到的应用程序指令
- ▶ 系统调用：等待和持续



中断、异常和系统调用

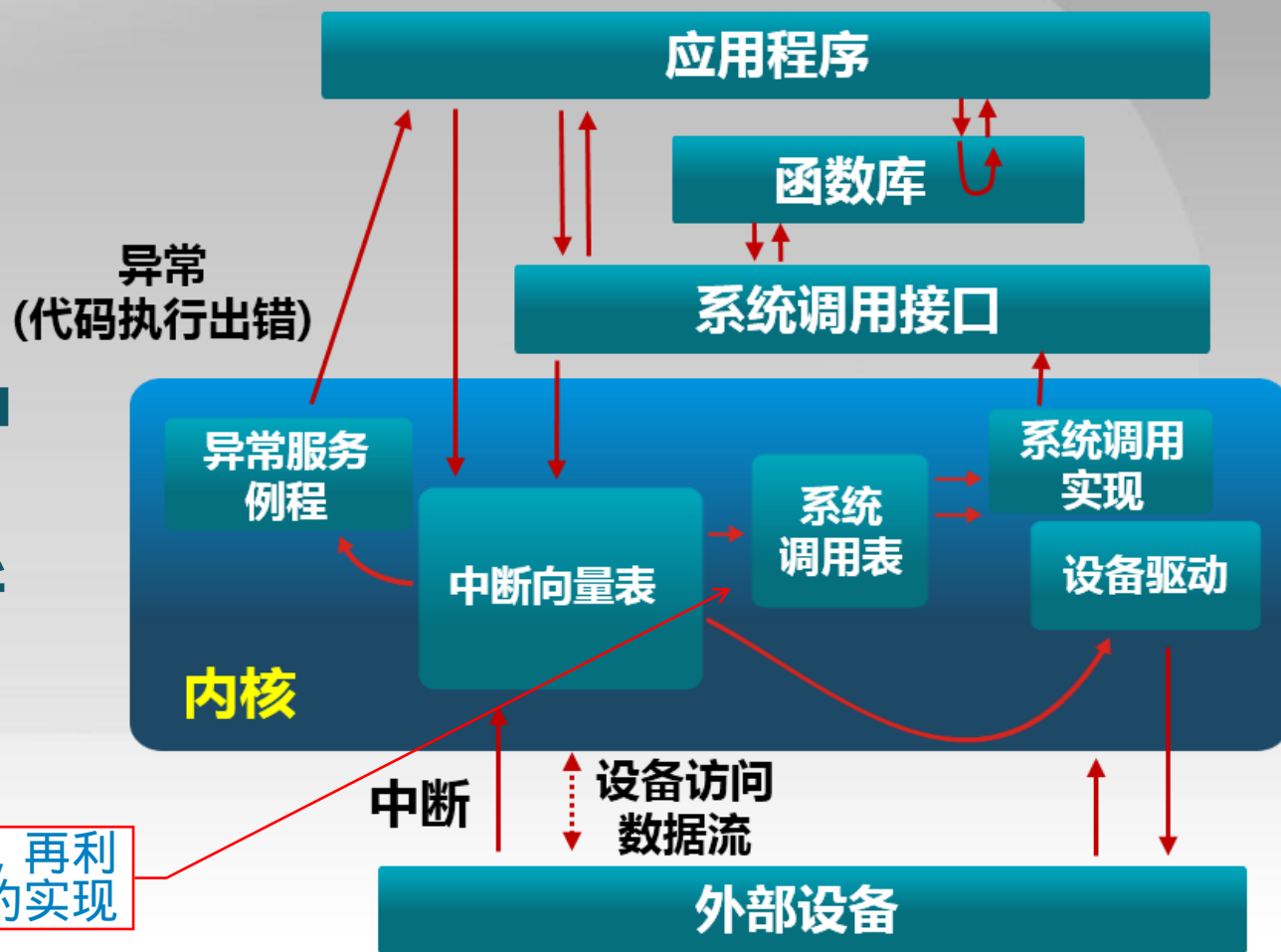
- 系统调用 (system call)
 - ▣ 应用程序**主动**向操作系统发出的服务请求
- 异常(exception)
 - ▣ 非法指令或者其他原因导致当前**指令执行失败**
(如：内存出错)后的处理请求
- 中断(hardware interrupt)
 - ▣ 来自硬件设备的处理请求

中断处理机制

硬件处理

■ 在CPU初始化时设置中断使能标志

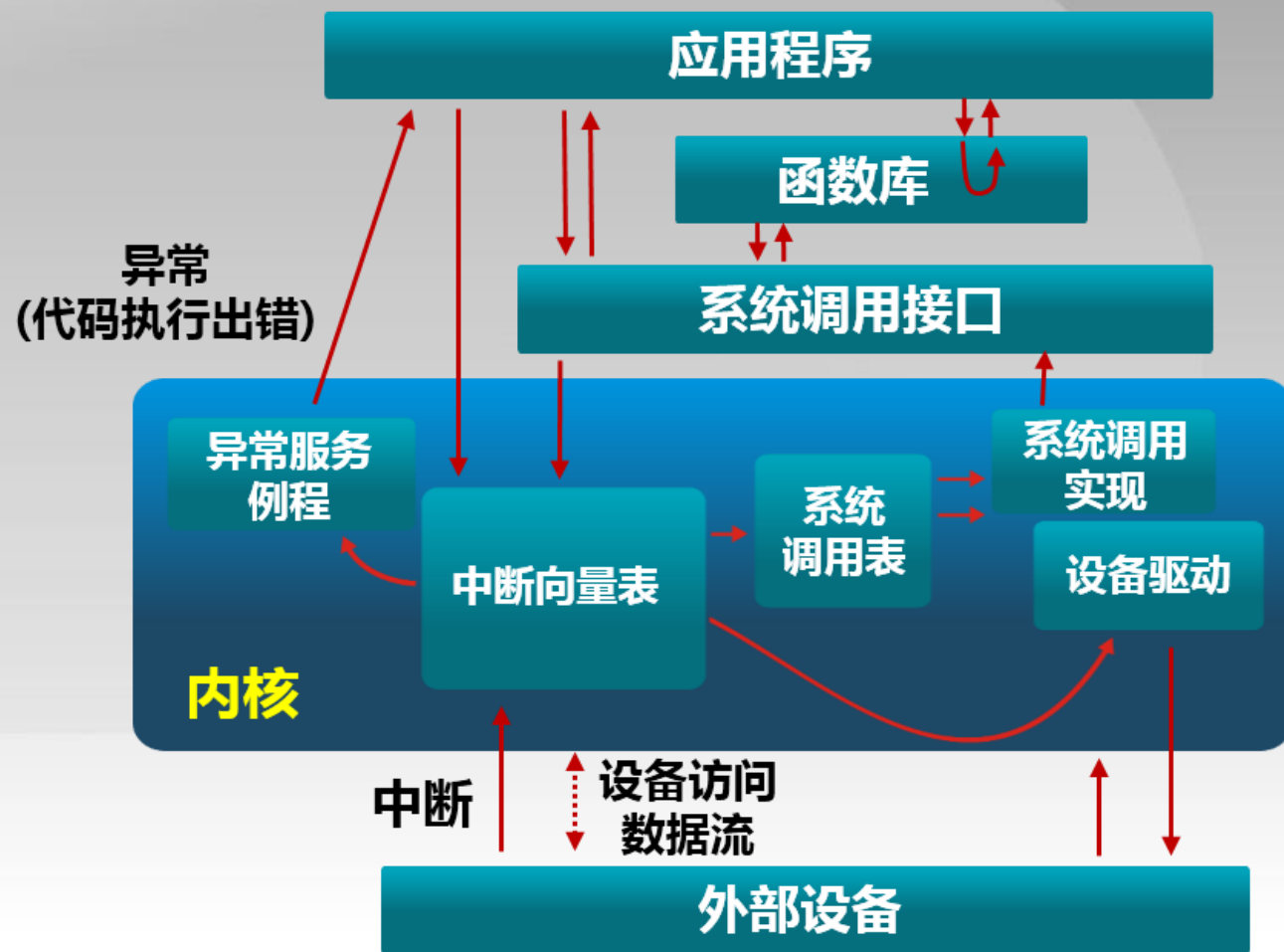
- ▣ 依据内部或外部事件设置中断标志
- ▣ 依据中断向量调用相应中断服务例程



中断和异常处理机制

软件

- ▣现场保存（CPU+编译器）
- ▣中断服务处理（服务例程）
- ▣清除中断标记（服务例程）
- ▣现场恢复（CPU+编译器）



中断嵌套

- 硬件中断服务例程可被打断
 - ▣ 不同硬件中断源可能在硬件中断处理时出现
 - ▣ 硬件中断服务例程中需要临时禁止中断请求
 - ▣ 中断请求会保持到CPU做出响应
- 异常服务例程可被打断
 - ▣ 异常服务例程执行时可能出现硬件中断
- 异常服务例程可嵌套
 - ▣ 异常服务例程可能出现缺页



操作系统

Operating System

标准C库的例子

- 应用程序调用printf() 时，会触发系统调用write()。



系统调用

- 操作系统服务的编程接口
- 通常由高级语言编写（C或者C++）
- 程序访问通常是通过高层次的API接口而不是直接进行系统调用
- 三种最常用的应用程序编程接口（API）
 - ▣ Win32 API 用于 Windows
 - ▣ POSIX API 用于 POSIX-based systems (包括UNIX, LINUX, Mac OS X的所有版本)
 - ▣ Java API 用于JAVA虚拟机(JVM)

ABI 和API的区别是？

系统调用的实现

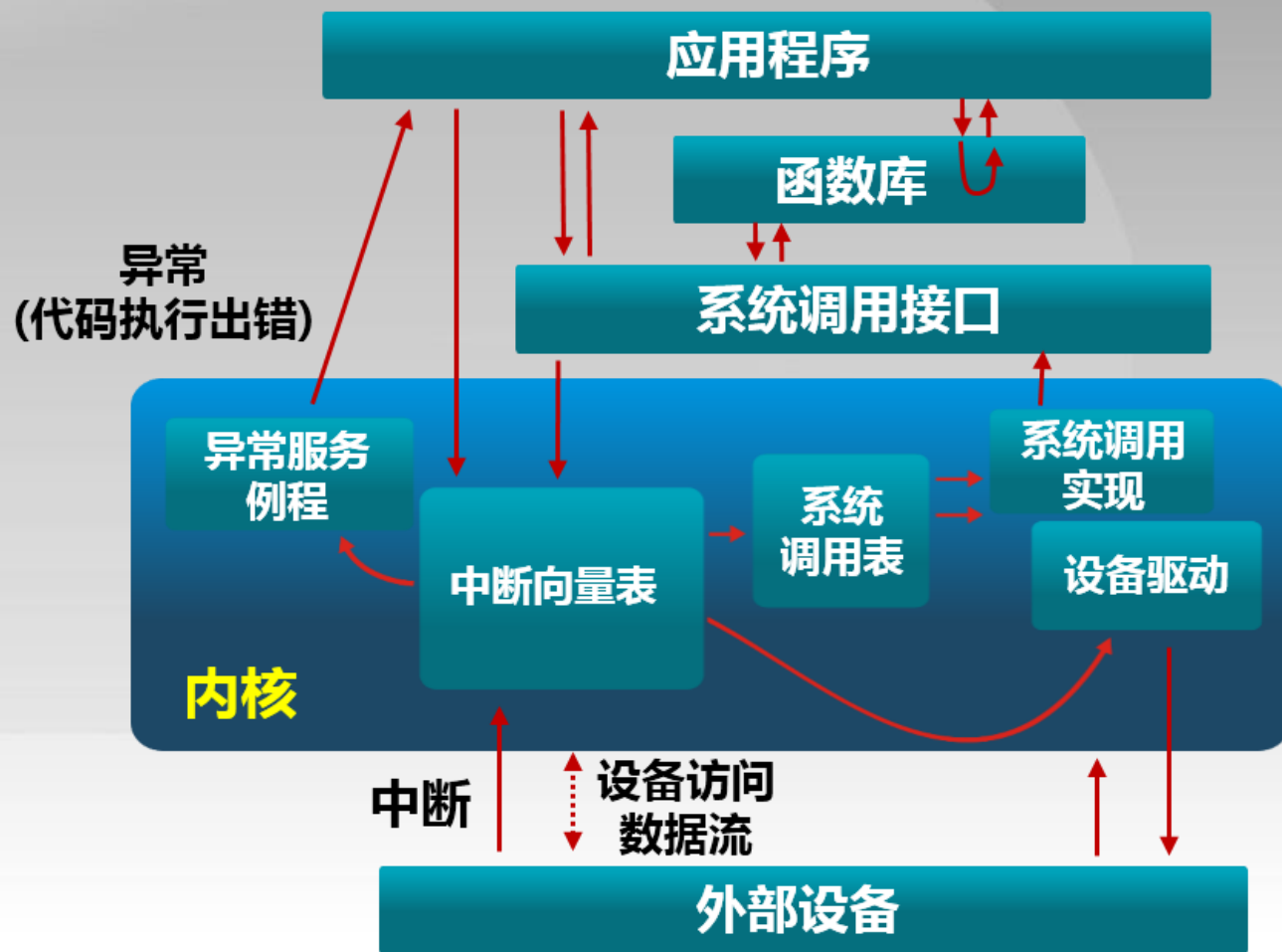
■ 每个系统调用对应一个系统调用号

- ▶ 系统调用接口根据系统调用号来维护表的索引

■ 系统调用接口调用内核态中的系统调用功能实现，并返回系统调用的状态和结果

■ 用户不需要知道系统调用的实现

- ▶ 需要设置调用参数和获取返回结果
- ▶ 操作系统接口的细节大部分都隐藏在应用编程接口后
 - 通过运行程序支持的库来管理



函数调用和系统调用的不同处

■ 系统调用

▣ INT和IRET指令用于系统调用

- 系统调用时，堆栈切换和特权级的转换

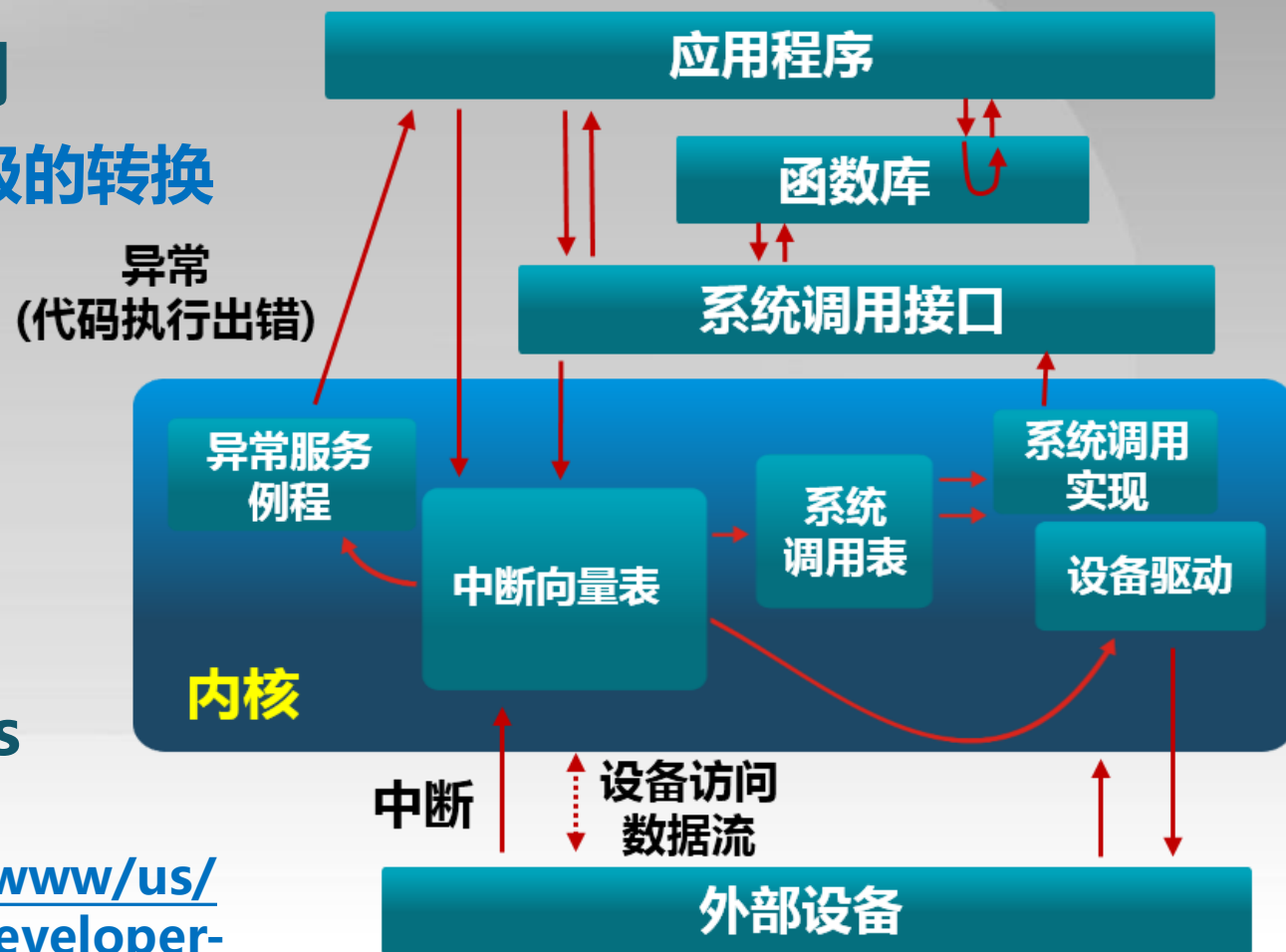
■ 函数调用

▣ CALL和RET用于常规调用

- 常规调用时没有堆栈切换

■ Intel 64 and IA-32 Architectures Software Developer

[Manuals](http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html)
<http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>



中断、异常和系统调用的开销

■ 超过函数调用

■ 开销：

▣ 引导机制

▣ 建立内核堆栈

第一次调用时需要建立内核堆栈

▣ 验证参数

参数的安全性需要验证

▣ 内核态映射到用户态的地址空间

- 更新页面映射权限

▣ 内核态独立地址空间

- TLB



操作系统

Operating System

系统调用示例

■ 文件复制过程中的系统调用序列

源文件

获取输入文件名
在屏幕显示提示
等待并接收键盘输入
获取输出文件名
在屏幕显示提示
等待并接收键盘输入
打开输入文件
如果文件不存在，出错退出
创建输出文件
如果文件存在，出错退出
循环
 读取输入文件
 写入输出文件
直到读取结束
关闭输出文件
在屏幕显示完成信息
正常退出

目标文件

```
// System call numbers
#define SYS_fork 1
#define SYS_exit 2
#define SYS_wait 3
#define SYS_pipe 4
#define SYS_write 5
#define SYS_read 6
#define SYS_close 7
#define SYS_kill 8
#define SYS_exec 9
#define SYS_open 10
#define SYS_mknod 11
#define SYS_unlink 12
#define SYS_fstat 13
#define SYS_link 14
#define SYS_mkdir 15
#define SYS_chdir 16
#define SYS_dup 17
#define SYS_getpid 18
#define SYS_sbrk 19
#define SYS_sleep 20
#define SYS_procmem 21
```

系统调用示例

- 在ucore中库函数read()的功能是读文件
 - ▣ user/libs/file.h: `int read(int fd, void * buf, int length)`
- 库函数read()的参数和返回值
 - ▣ int fd—文件句柄
 - ▣ void * buf—数据缓冲区指针
 - ▣ int length—数据缓冲区长度
 - ▣ int return_value:返回读出数据长度
- 库函数read()使用示例
 - ▣ in sfs_filetest1.c: `ret = read(fd, data, len);`

系统调用库接口示例

```
sfs_filetest1.c: ret=read(fd,data,len);
```

```
.....
```

8029a1:	8b 45 10	mov	0x10(%ebp),%eax
8029a4:	89 44 24 08	mov	%eax,0x8(%esp)
8029a8:	8b 45 0c	mov	0xc(%ebp),%eax
8029ab:	89 44 24 04	mov	%eax,0x4(%esp)
8029af:	8b 45 08	mov	0x8(%ebp),%eax
8029b2:	89 04 24	mov	%eax,(%esp)
8029b5:	e8 33 d8 ff ff	call	8001ed <read>

```
syscall(int num, ...) {
```

```
...
```

```
    asm volatile (
```

```
        "int %1;"
        : "=a" (ret)
        : "i" (T_SYSCALL),
          "a" (num),
          "d" (a[0]),
          "c" (a[1]),
          "b" (a[2]),
          "D" (a[3]),
          "S" (a[4])
        : "cc", "memory");
```

```
    return ret;
```

前半部分工作：压栈

ucore系统调用read(fd, buffer, length)的实现

1. kern/trap/trapentry.S: alltraps()
2. kern/trap/trap.c: trap()
tf->trapno == T_SYSCALL
3. kern/syscall/syscall.c: syscall()
tf->tf_regs.reg_eax == SYS_read
4. kern/syscall/syscall.c: sys_read()
从 tf->sp 获取 fd, buf, length
5. kern/fs/sysfile.c: sysfile_read()
读取文件
6. kern/trap/trapentry.S: trapret()



操作系统

Operating System