从机制与策略探究Linux内核设计之道

嵌入式系统联谊会
http://www.esbf.org.cn

西安邮电学院　陈莉君

形而上谓之道

形而下谓之器

—《易经》

所见的是暂时的

所不见的是永远的

—《圣经》

文、史、哲

文学--使看不见的东西被看见

哲学-----迷宫中望见星空

史学----沙漠玫瑰的开放

摘自龙应台的《百年思索》

《天问》

天何所沓　十二焉分

日月安属　列星安陈

何阖而晦　何开而明

角宿未旦　曜灵安藏

--屈原

做学问要在不疑处有疑

大胆的假设，小心的求证

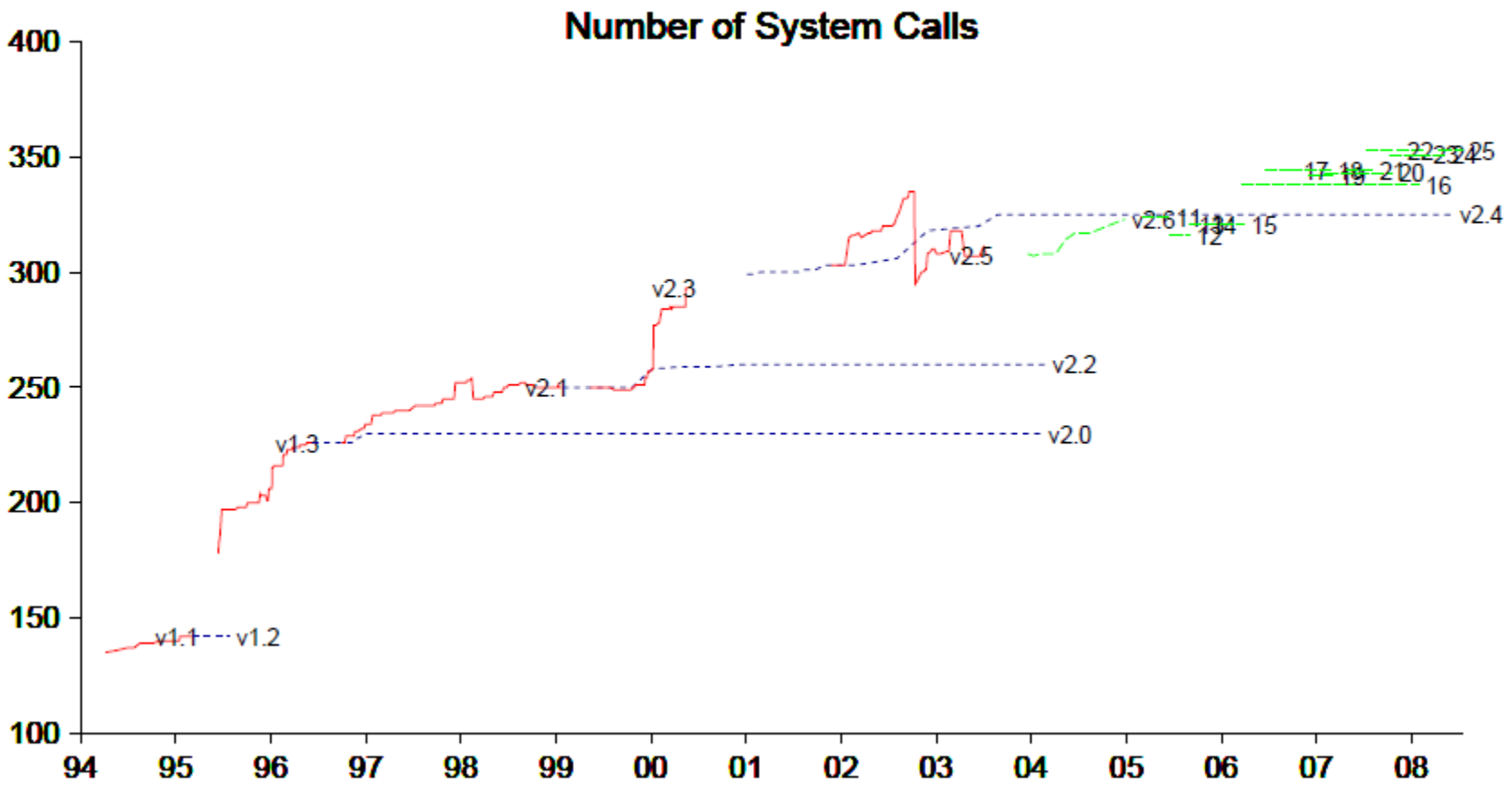
--胡适

Linux内核版本的历史演变

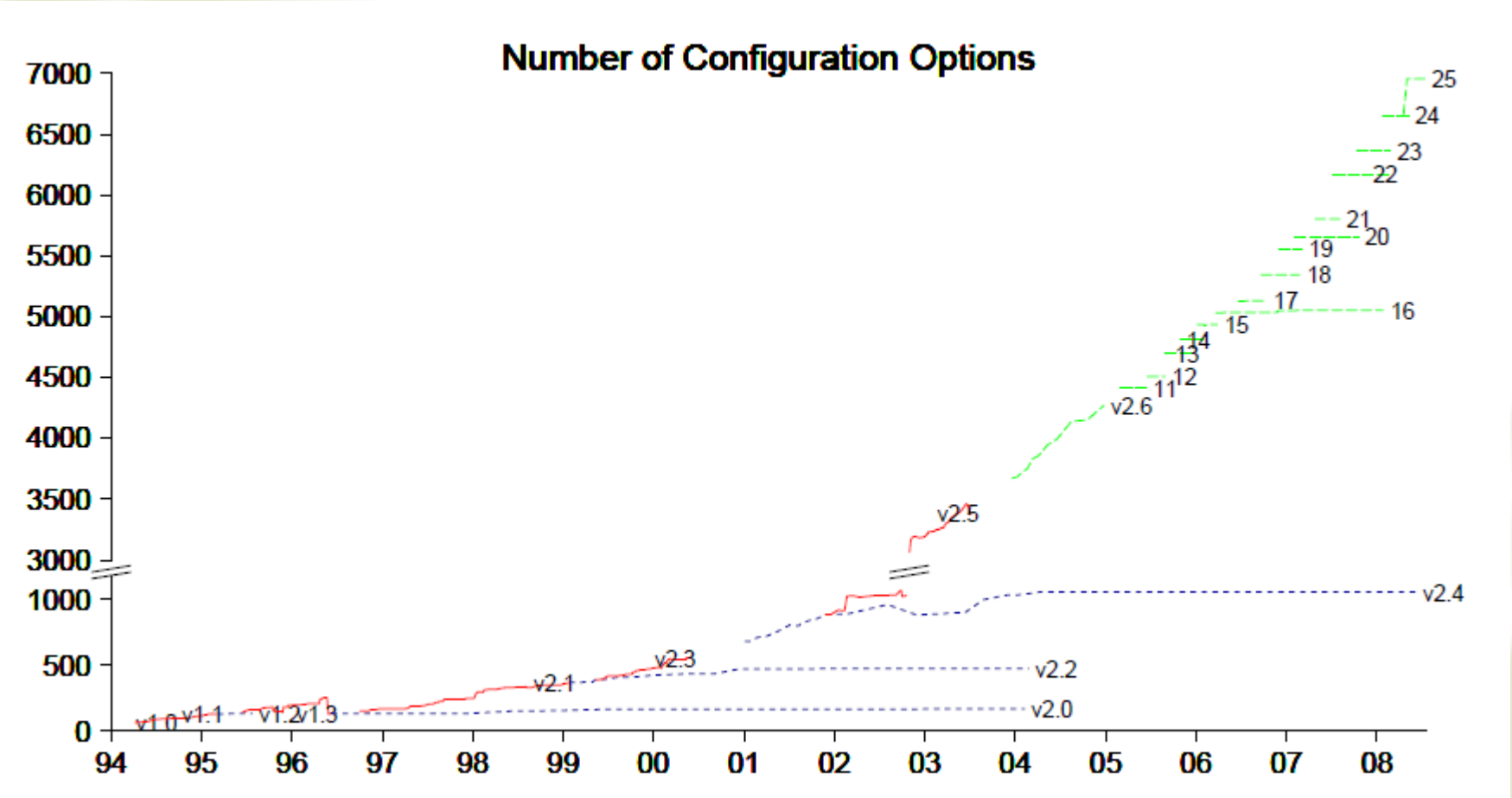
Linux内核一路走来

- Linux内核版本达800多个
- 系统调用数随版本升级增加
- 内核配置选项随版本大幅增加
- 代码行数有规律递增
- 函数的平均复杂度降低，但函数数量增加

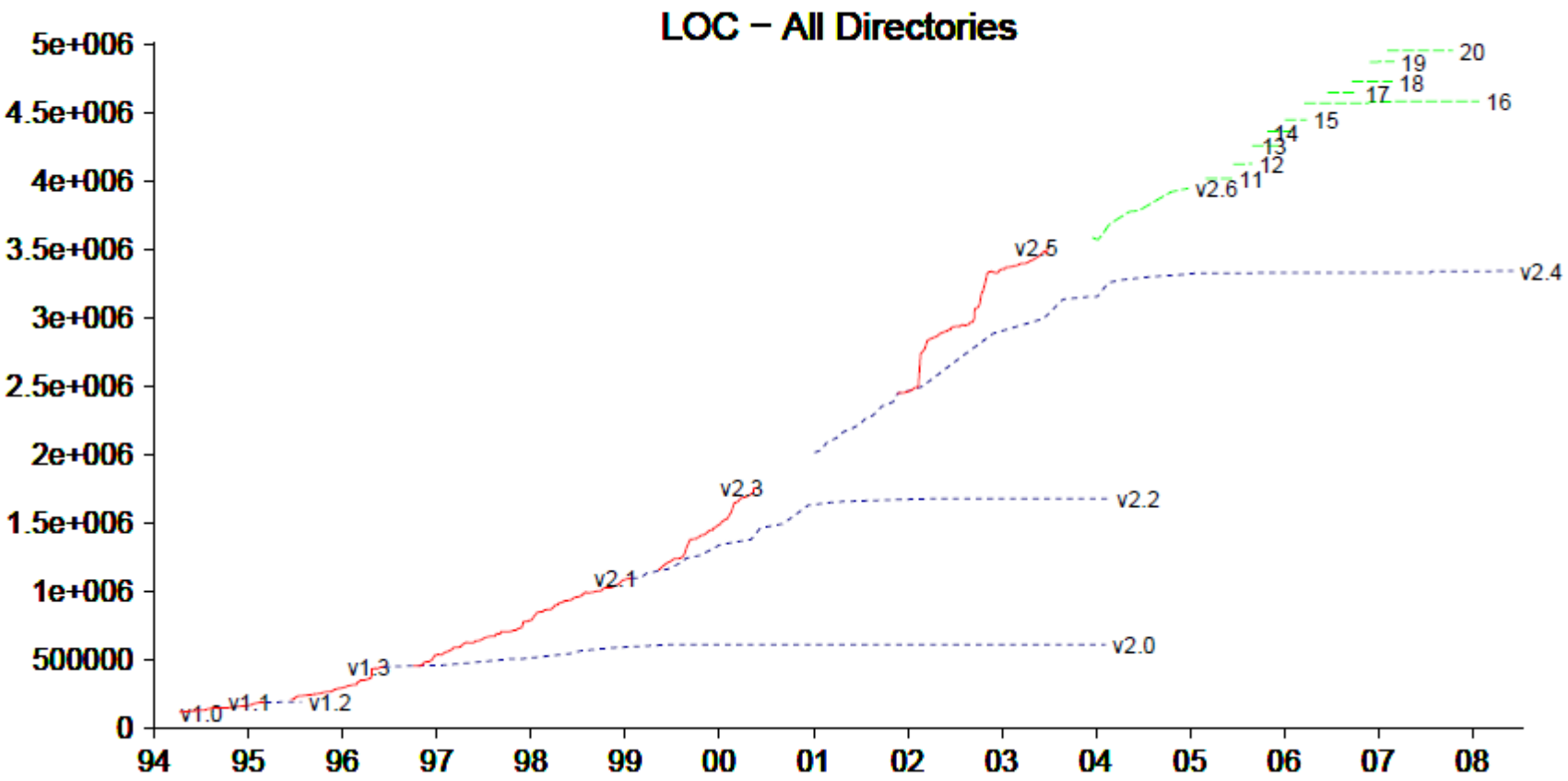
系统调用数的变化


Number of System Calls
400
350
300
250
200
150
100
94
95
96
97
98
99
00
01
02
03
04
05
06
07
08
v1.1
v1.2
v1.3
v2.0
v2.1
v2.2
v2.3
v2.4
v2.5
v2.6
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

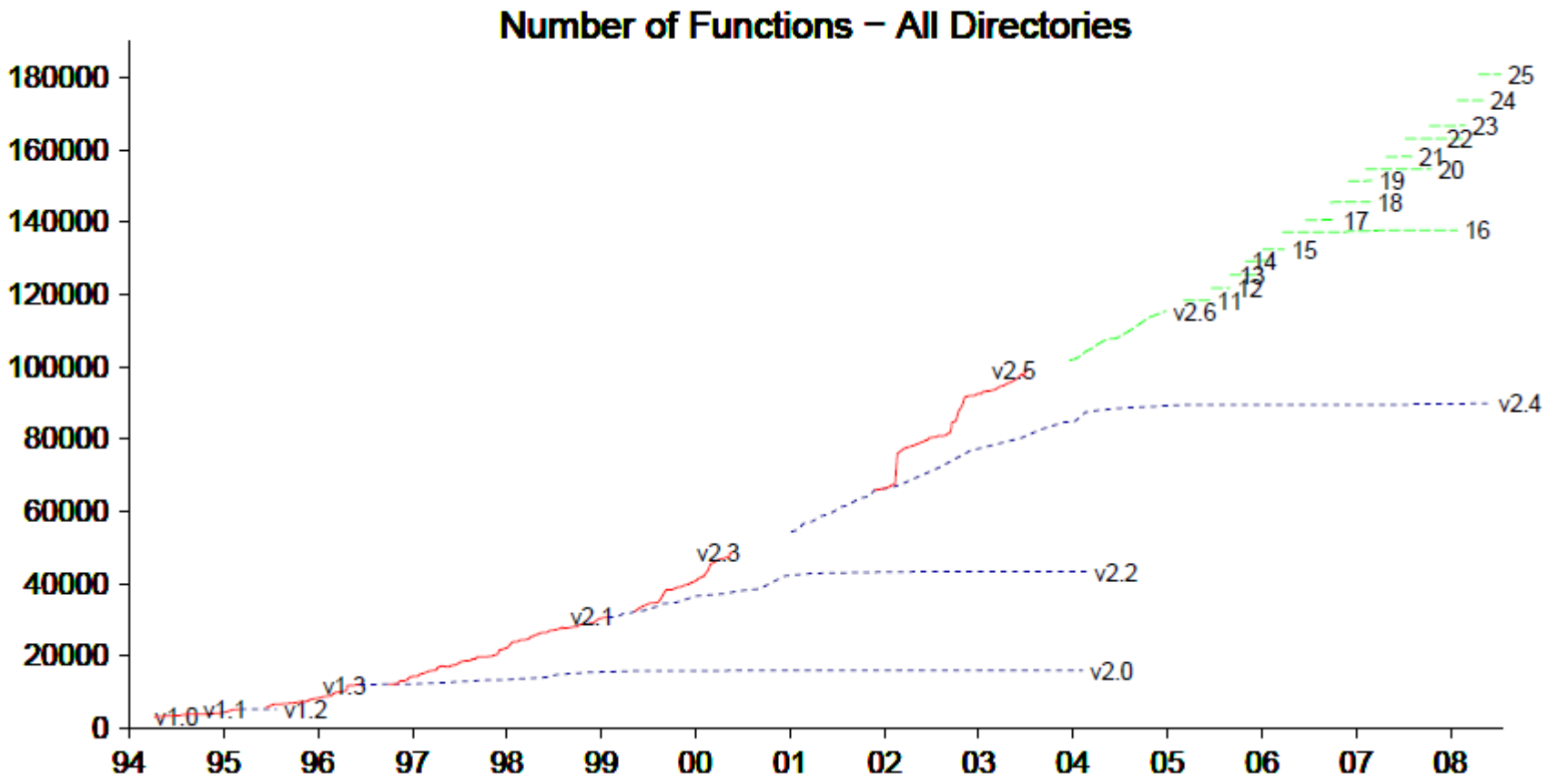
内核配置选项的变化


Number of Configuration Options
7000
6500
6000
5500
5000
4500
4000
3500
3000
1000
500
0
94
95
96
97
98
99
00
01
02
03
04
05
06
07
08
v1.0
v1.1
v1.2/1.3
v2.0
v2.1
v2.2
v2.3
v2.4
v2.5
v2.6
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

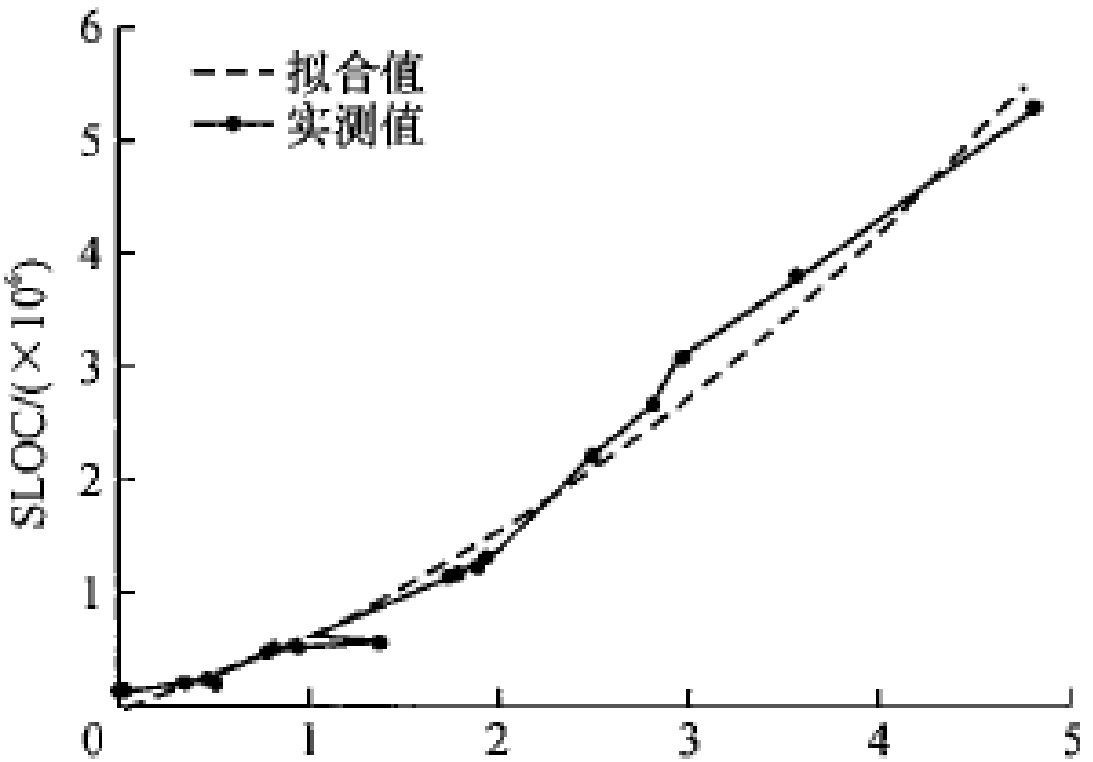
代码量的变化


LOC – All Directories
5e+006
4.5e+006
4e+006
3.5e+006
3e+006
2.5e+006
2e+006
1.5e+006
1e+006
500000
0
94
95
96
97
98
99
00
01
02
03
04
05
06
07
08
v1.0
v1.1
v1.2
v1.3
v2.0
v2.1
v2.2
v2.3
v2.4
v2.5
v2.6
11
12
13
14
15
16
17
18
19
20

函数个数的变化

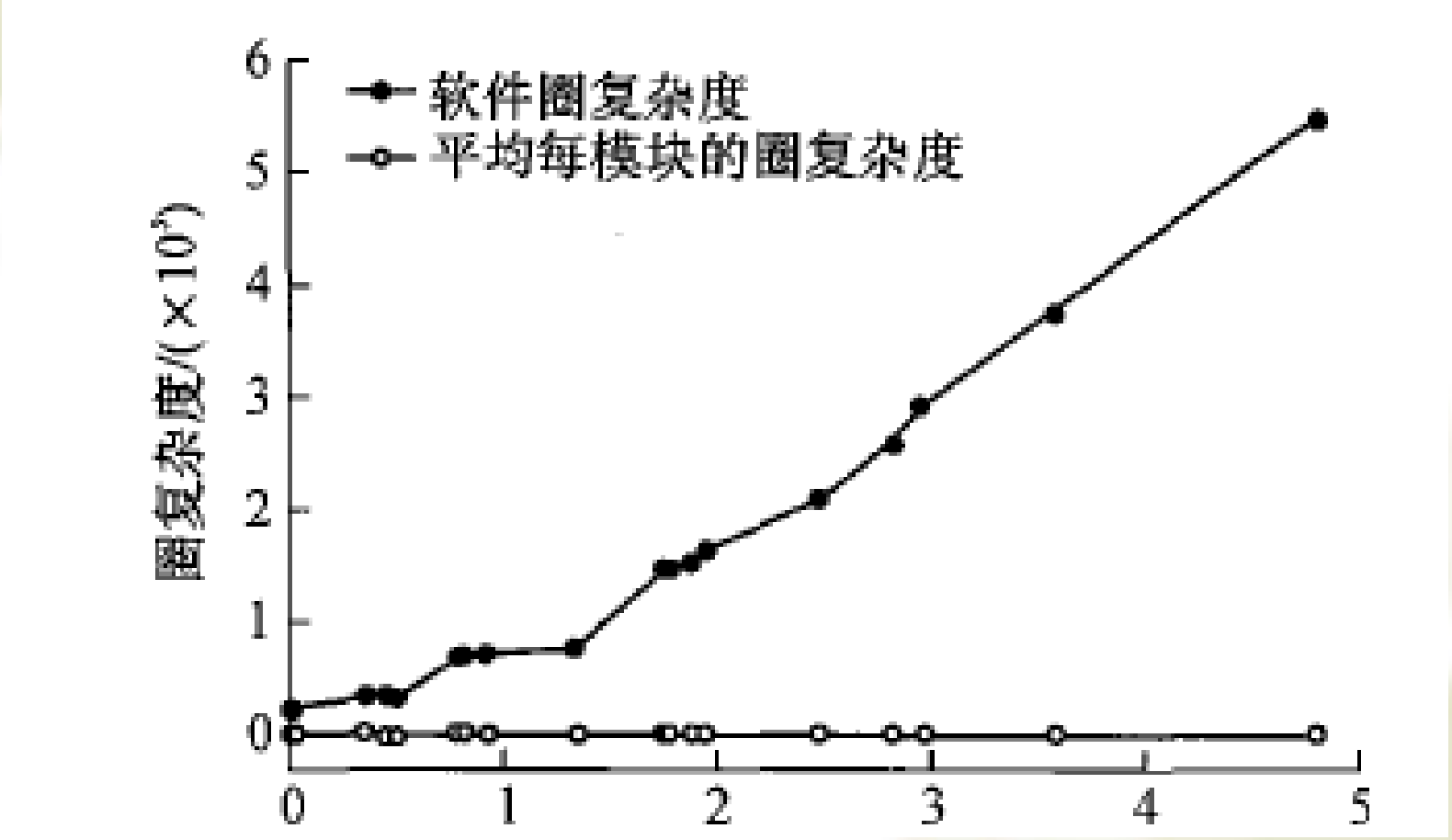

Number of Functions – All Directories
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
94
95
96
97
98
99
00
01
02
03
04
05
06
07
08
v1.0
v1.1
v1.2
v1.3
v2.0
v2.1
v2.2
v2.3
v2.4
v2.5
v2.6
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

内在规律-SLOC呈线性趋势

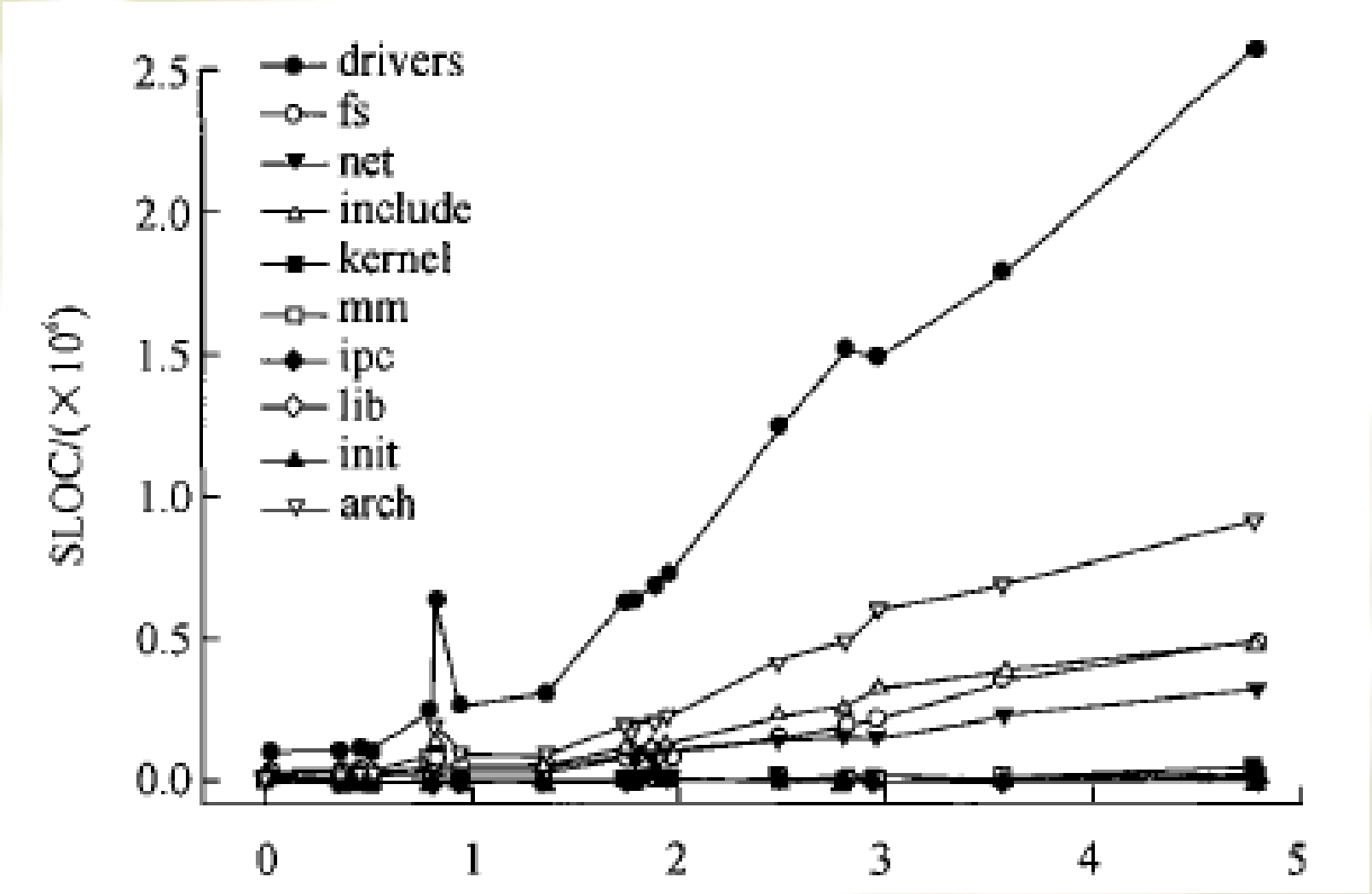


平均每个主要版本增加4万多行源代码

内在规律-圈复杂度



10个子系统的SLOC演化曲线



从SLOC所看到的

- Linux 的体系结构相对稳定
- 系统结构和子系统数变化不大，平均每模块的圈复杂度呈下降趋势.但系统整体规模和复杂性分别呈超线性和接近线性增长趋势。
- drivers 和arch 等子系统的快速变化引起是引起系统复杂性增加的主因
- Linux 演化的主要推动力是系统新特性新功能的增强，以适应系统资源的进化 。
- ....

内核演变引发的思考

- Linux内核演变的规律是什么？
- 在错综复杂的内核中最根本的是什么？
- 多视角的思维

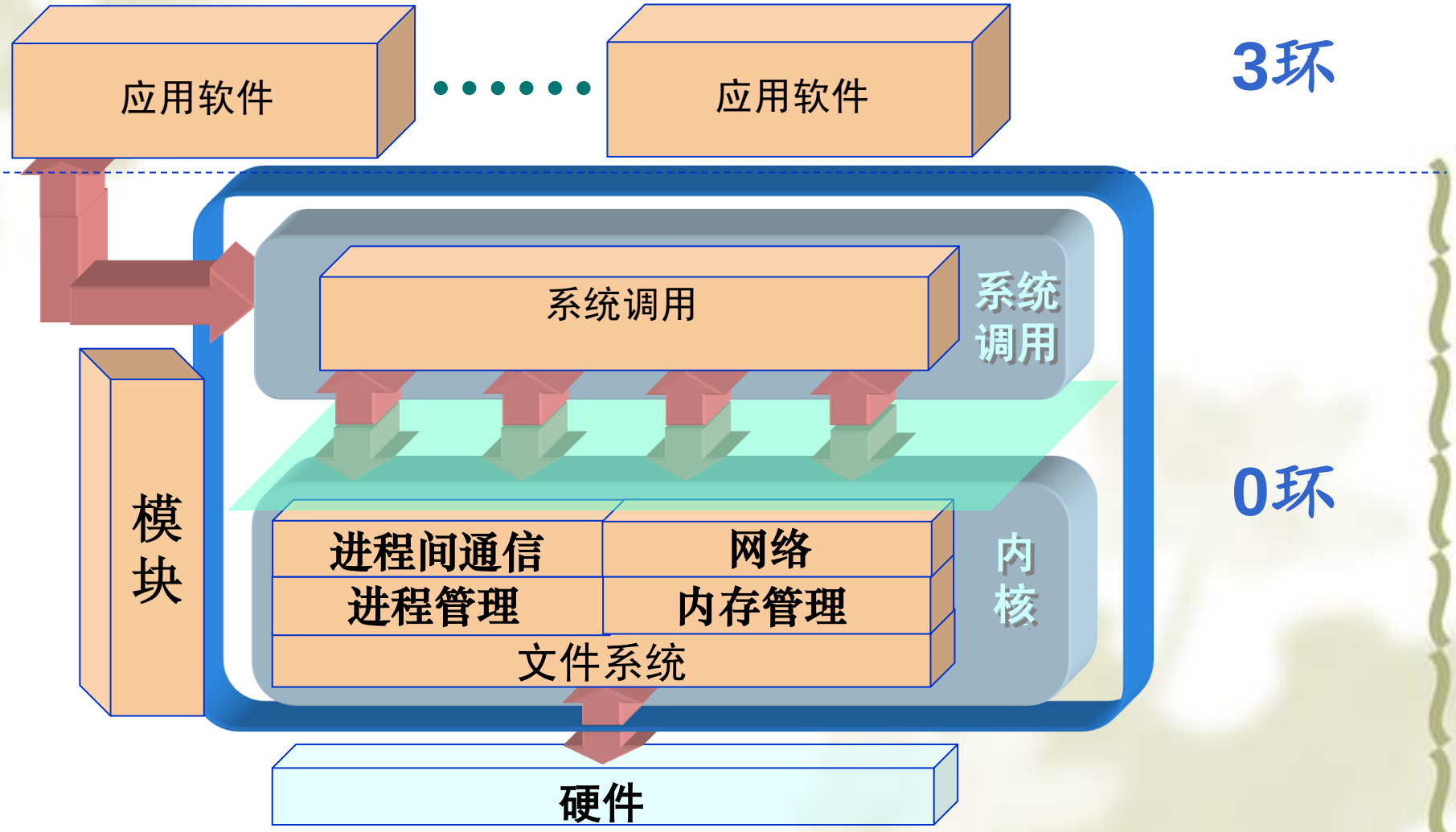
Unix/Linux设计理念

- 提供机制而不是策略
- 机制-提供什么样的功能
- 策略-怎样实现这些功能

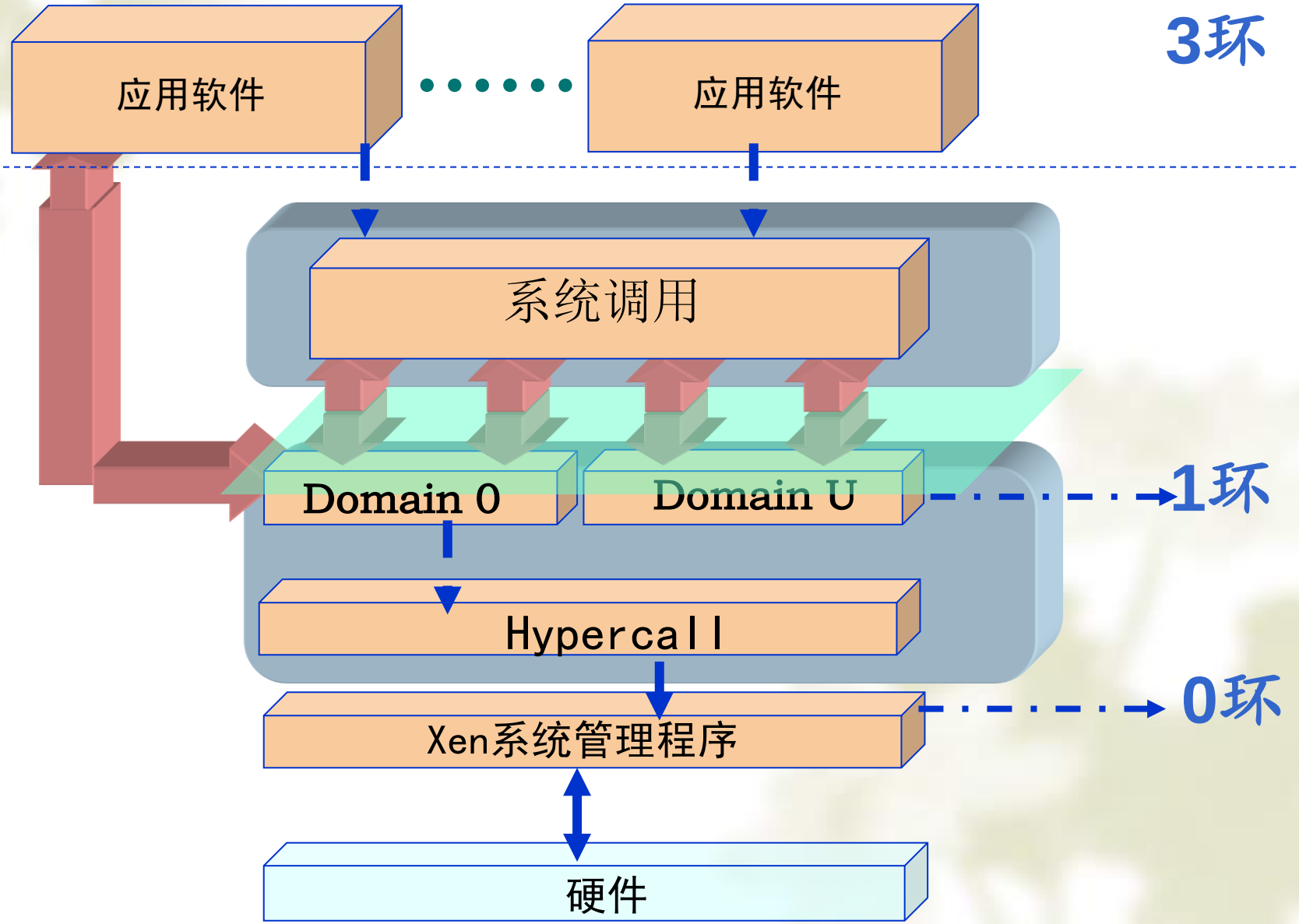
机制与策略

- 如果说机制是一种框架，那么，策略就是填充框架的一个个具体实体。
- 机制提供的是一种开放而宽松的环境，而策略就是在这个环境下赖以生存的生命个体。

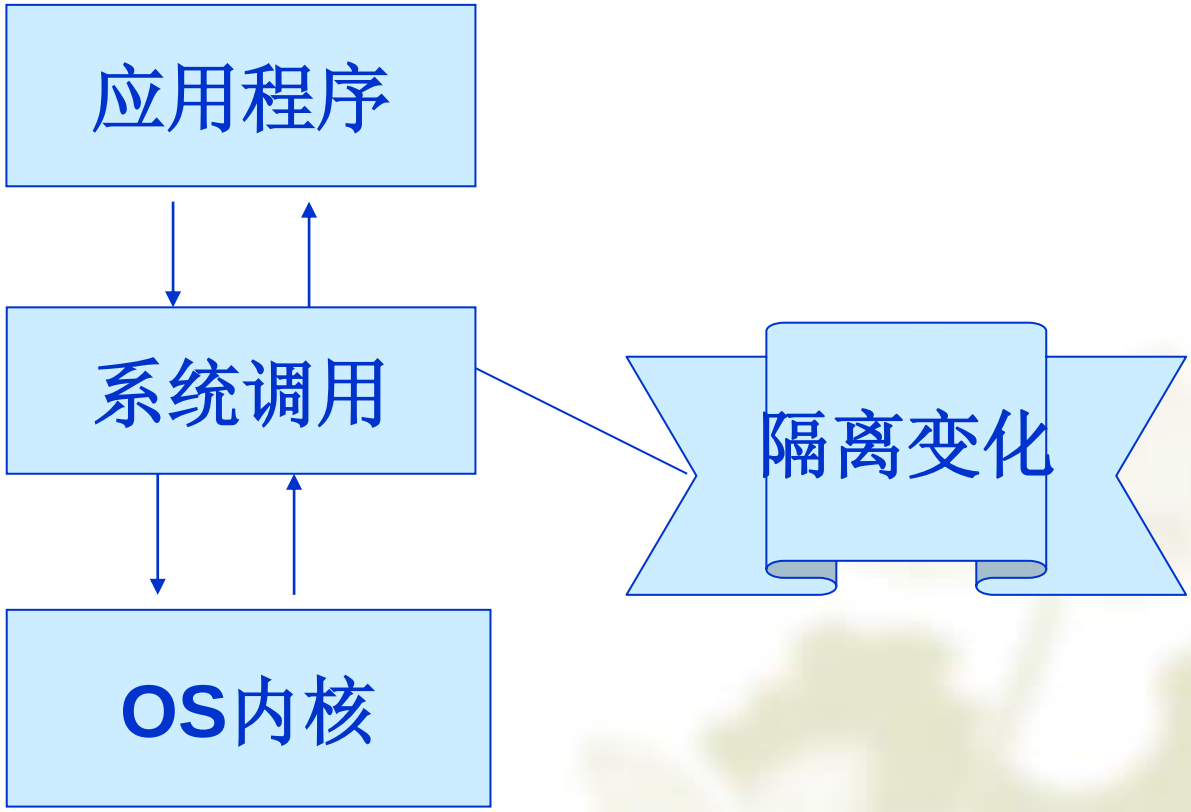
Linux系统整体结构



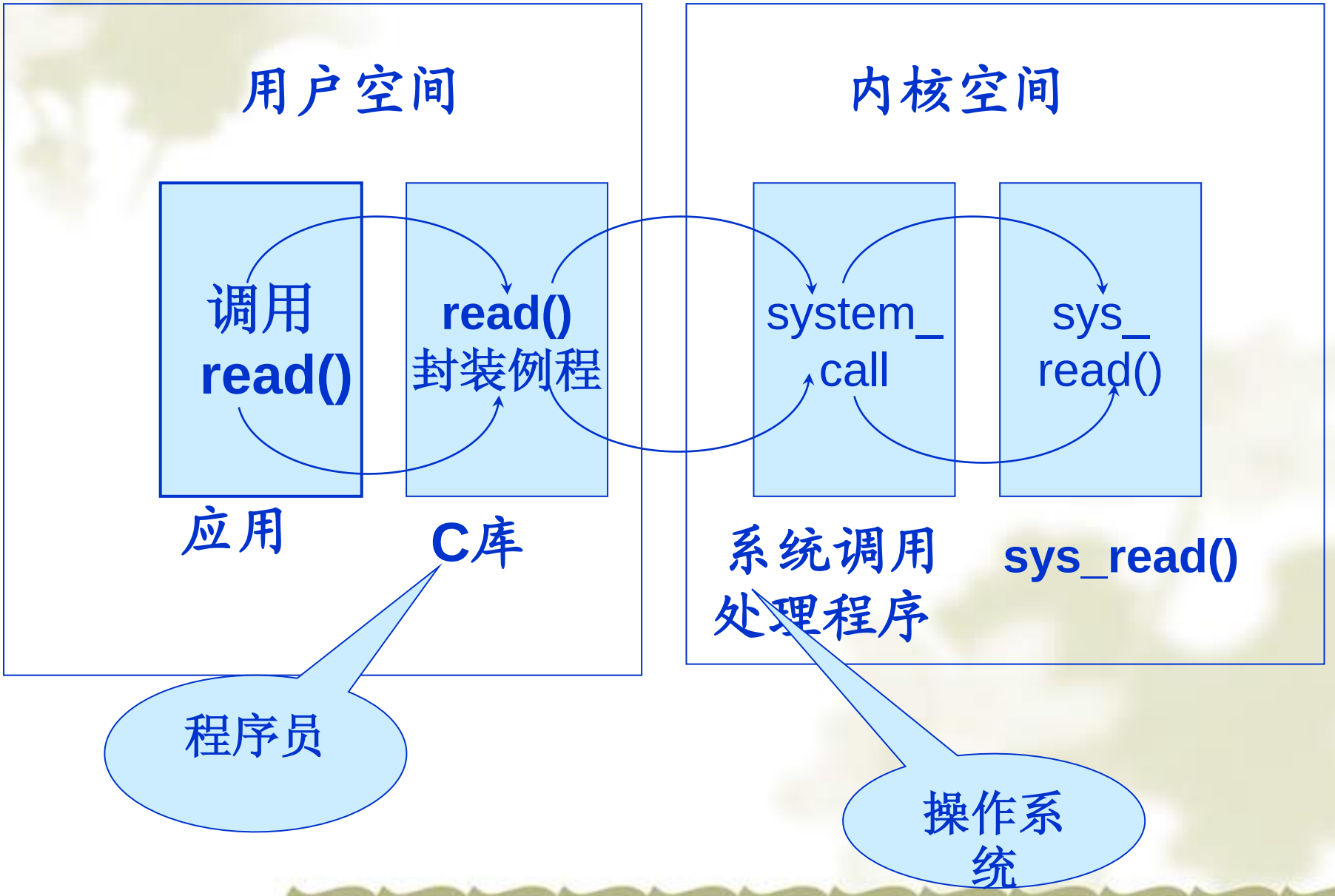
Xen体系结构


应用软件
应用软件
3环
系统调用
Domain 0
Domain U
1环
Hypercall
0环
Xen系统管理程序
硬件

隔离变化-系统调用机制



一个系统调用的执行机制



一个系统调用的执行机制

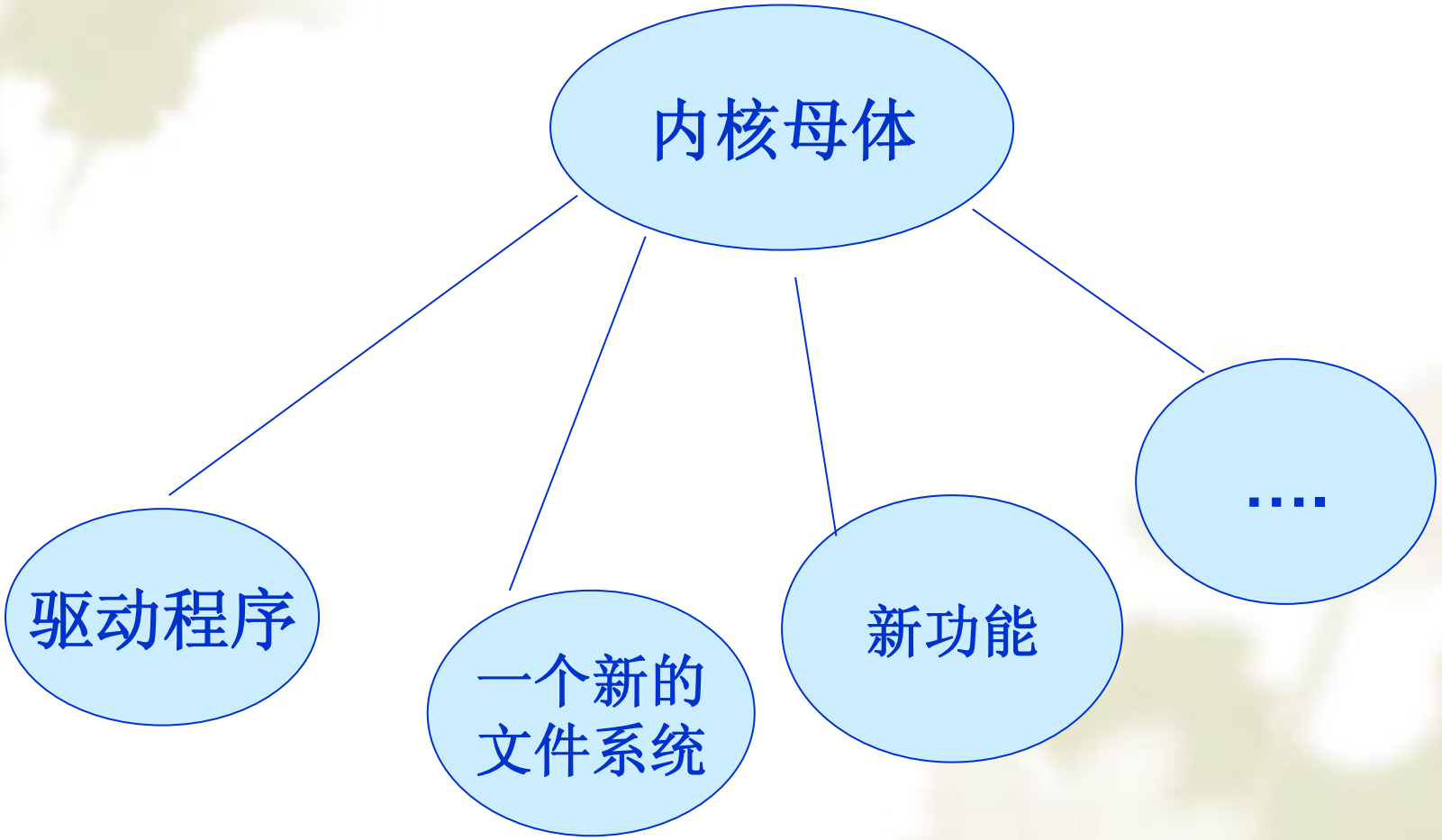
◆从程序的角度看，实际上不直接与系统调用打交道，而是跟API打交道，API是对系统调用的封装。

◆库函数以及应用程序怎样使用系统调用，内核并不关心，内核只跟系统调用打交道，而且是实现系统调用的大场景。

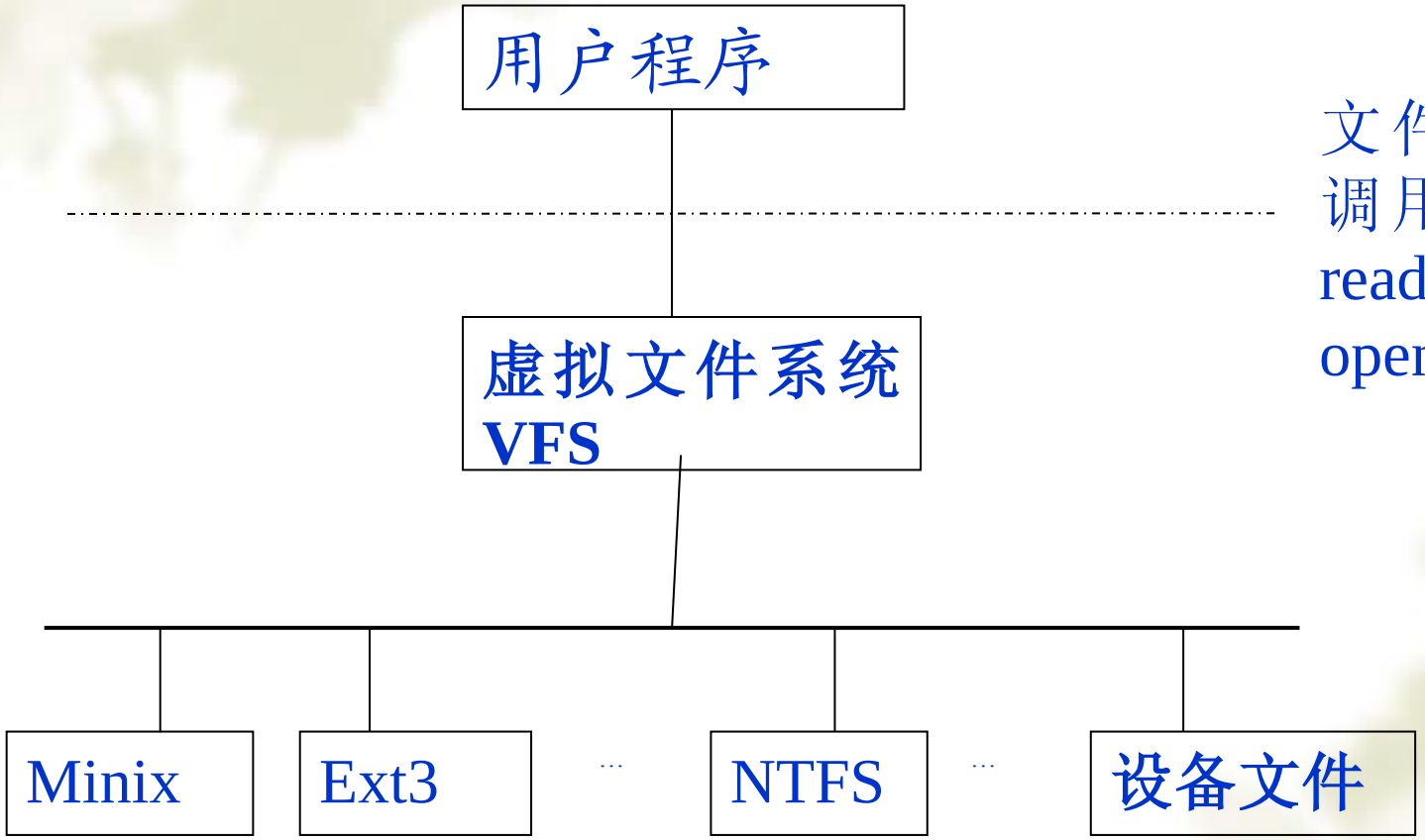
◆系统调用抽象出了用于完成某种特点目的的函数，至于这些函数怎么用不是内核关心的。

◆用户空间的程序无法直接执行内核代码。它们不能直接调用内核空间中的函数，因为内核驻留在受保护的地址空间上。如果进程可以直接在内核的地址空间上读写的话，系统的安全性和稳定性将不复存在。

应对变化-模块机制



隐藏差异-虚拟文件系统机制



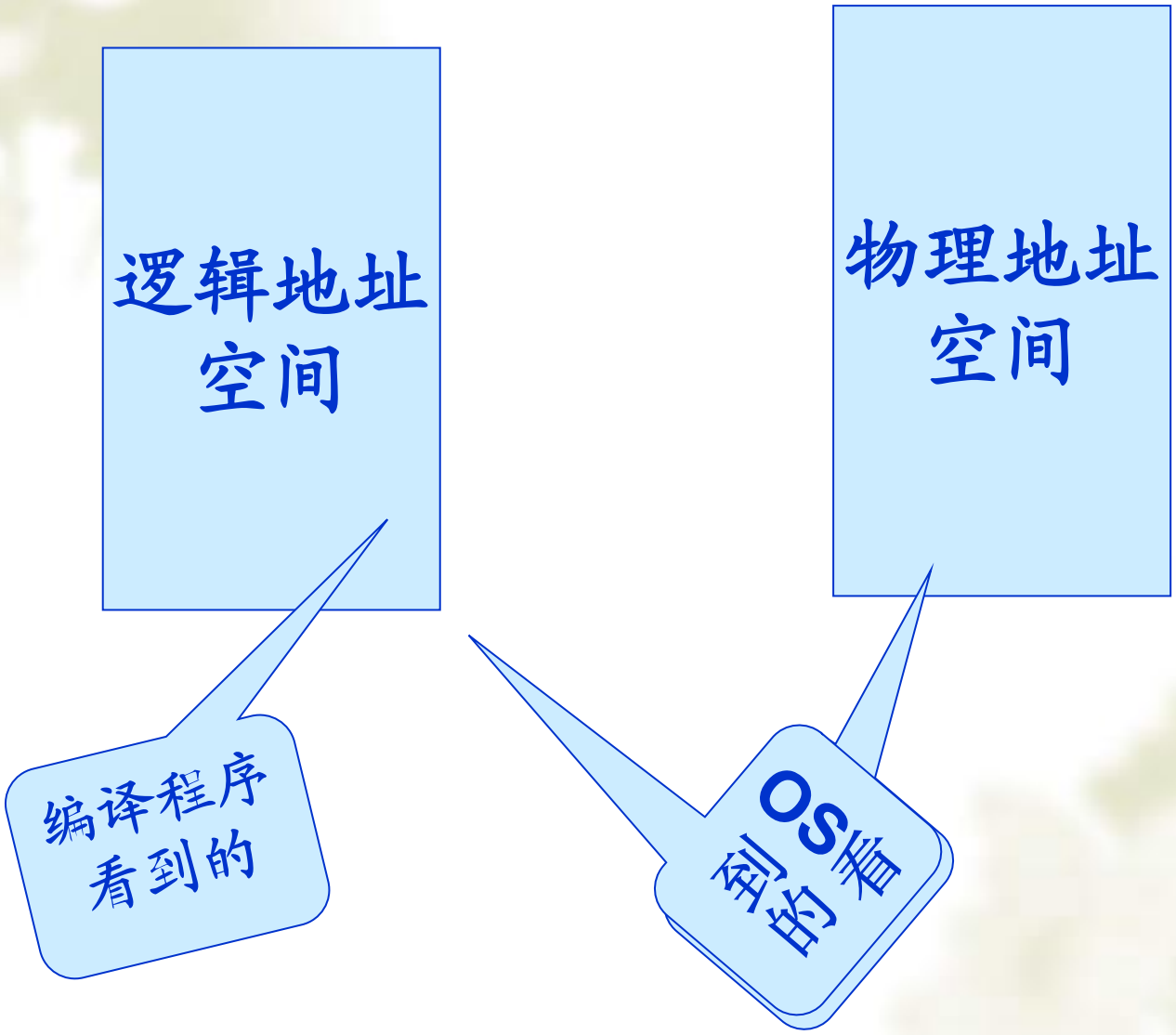
文件系统的系统调用接口，包括read()、write()、open()、close()等

VFS的机制

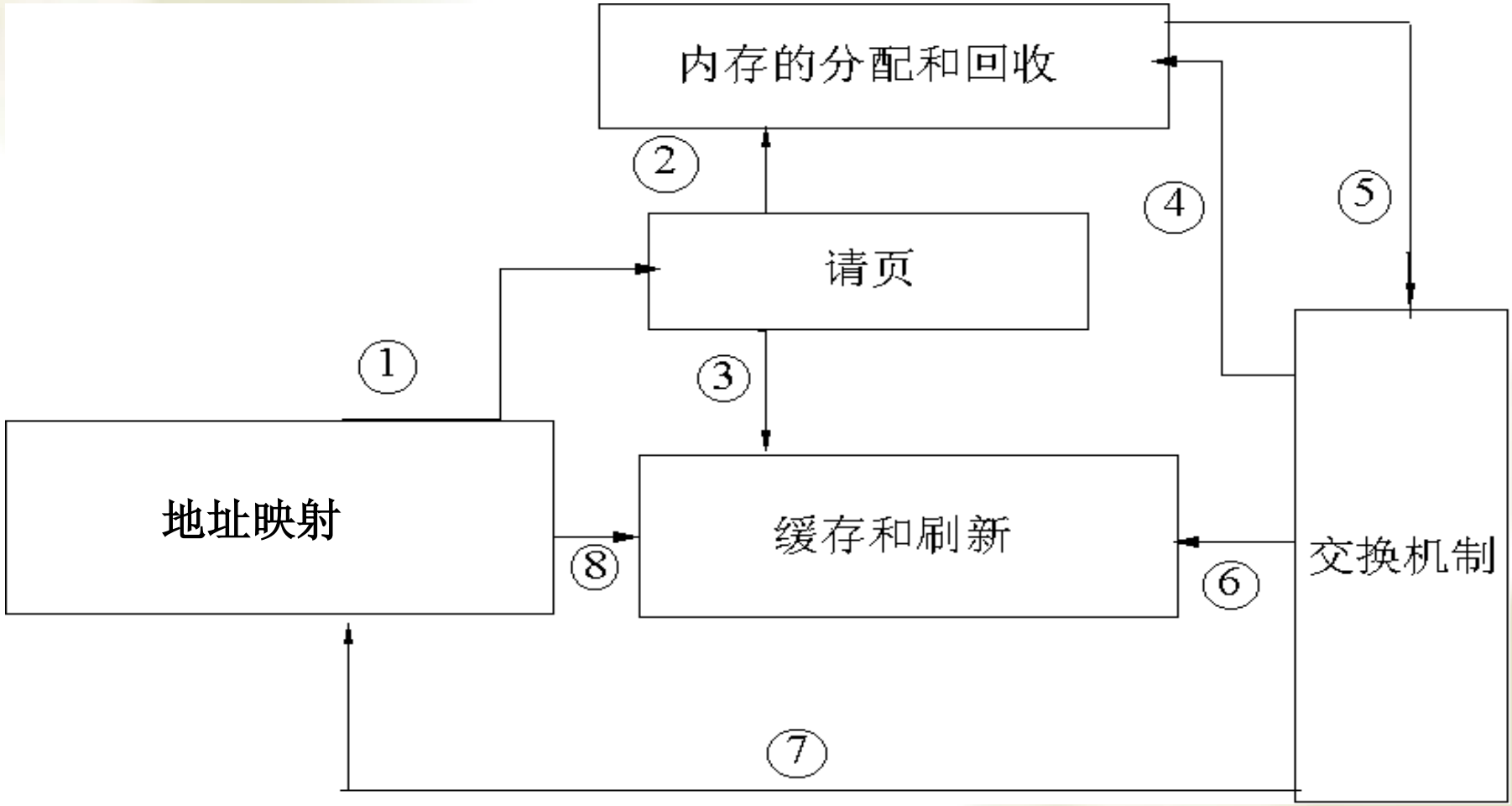
- 不同文件系统，不同驱动程序都有统一的接口
- 用户程序不关心具体文件系统的实现细节，使用抽象、统一、虚拟的界面

参见“Linux内核之旅”上关于Linux文件系统的两篇文章

内存管理机制



虚存管理机制



OS的本质是什么

- 管理者-管理软硬件资源，提高效率
- 服务者-提供各种服务，方便用户
- 归根结底是执行者

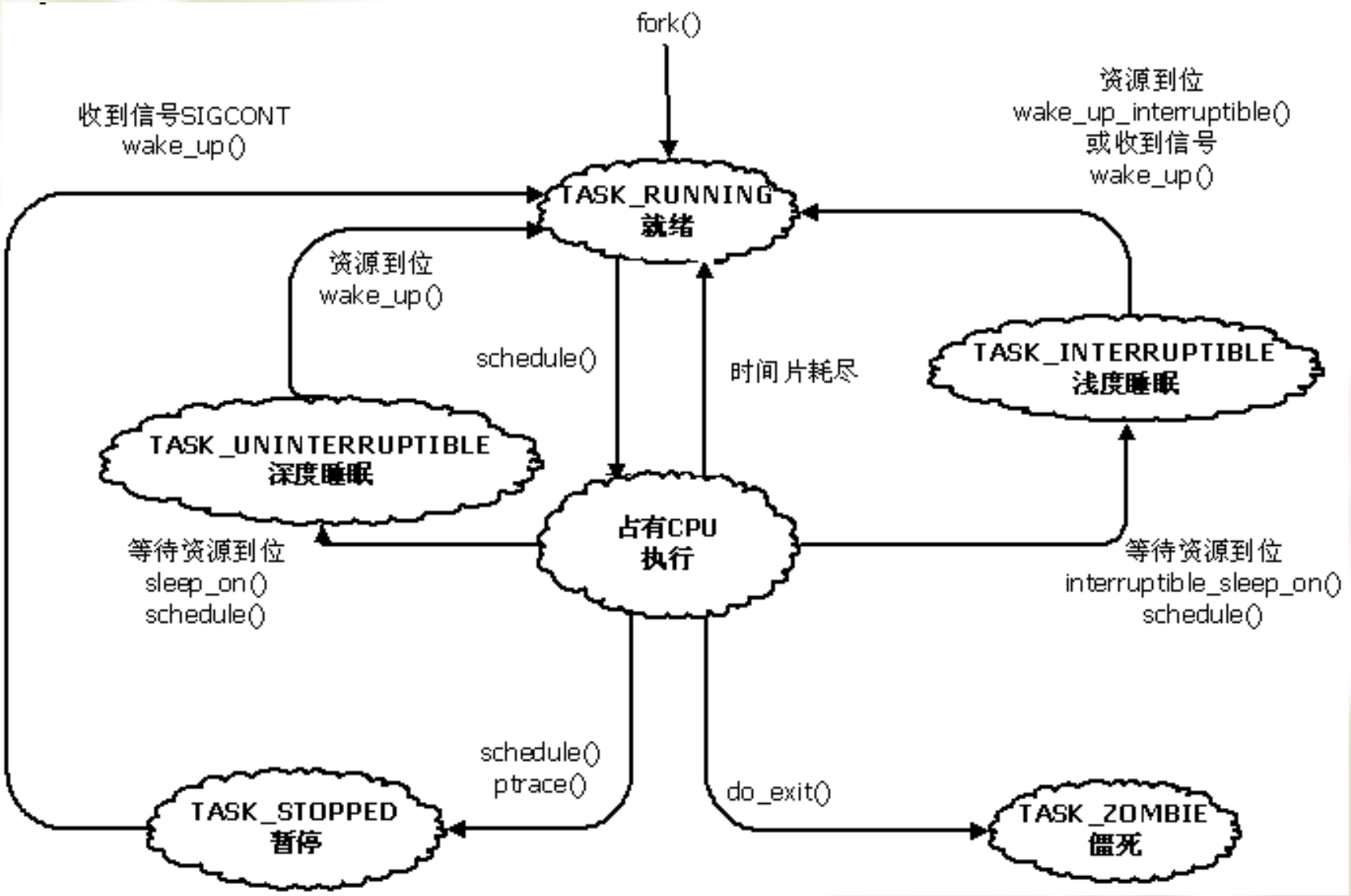
OS的执行机制

- 执行用户程序-为用户服务;
- 响应中断-为外设服务;
- 执行系统调用-解放程序员（因为程序员不再与繁杂的硬件打交道）
- 执行内核线程-为操作系统自身服务

OS的执行机制

- 执行用户程序-进程机制;
- 响应中断-中断机制;
- 执行系统调用-软中断机制
- 执行内核线程-线程机制

进程/线程的执行机制


fork()
收到信号SIGCONT
wake_up()
资源到位
wake_up_interruptible()
或收到信号
wake_up()
TASK_RUNNING
就绪
资源到位
wake_up()
schedule()
时间片耗尽
TASK_INTERRUPTIBLE
浅度睡眠
TASK_UNINTERRUPTIBLE
深度睡眠
占有CPU
执行
等待资源到位
sleep_on()
schedule()
等待资源到位
interruptible_sleep_on()
schedule()
schedule()
ptrace()
do_exit()
TASK_STOPPED
暂停
TASK_ZOMBIE
僵死

中断机制

- 当新的设备引入新类型的中断时，CPU和操作系统不用关注如何处理它。
- os负责提供接口，让用户通过该接口注册根据设备具体功能而编制的中断服务程序。
- 如果用户注册了对应于一个中断的服务程序，那么CPU就会在该中断到来时调用用户注册的服务程序。
- 在中断来临时系统需要如何操作硬件、如何实现硬件功能这部分工作就完全独立于CPU架构和操作系统的设计。

参见Linux内核之旅上中断趣味谈

中断机制的实施

- Linux把系统调用、异常和外部中断纳入统一框架处理。
- 外部中断分为上下两半部分处理。
- 上半部分就是通常的中断处理过程
- 下班部分有多种处理机制： softirq，tasklet，工作队列

中断下半部机制的实施-tasklet

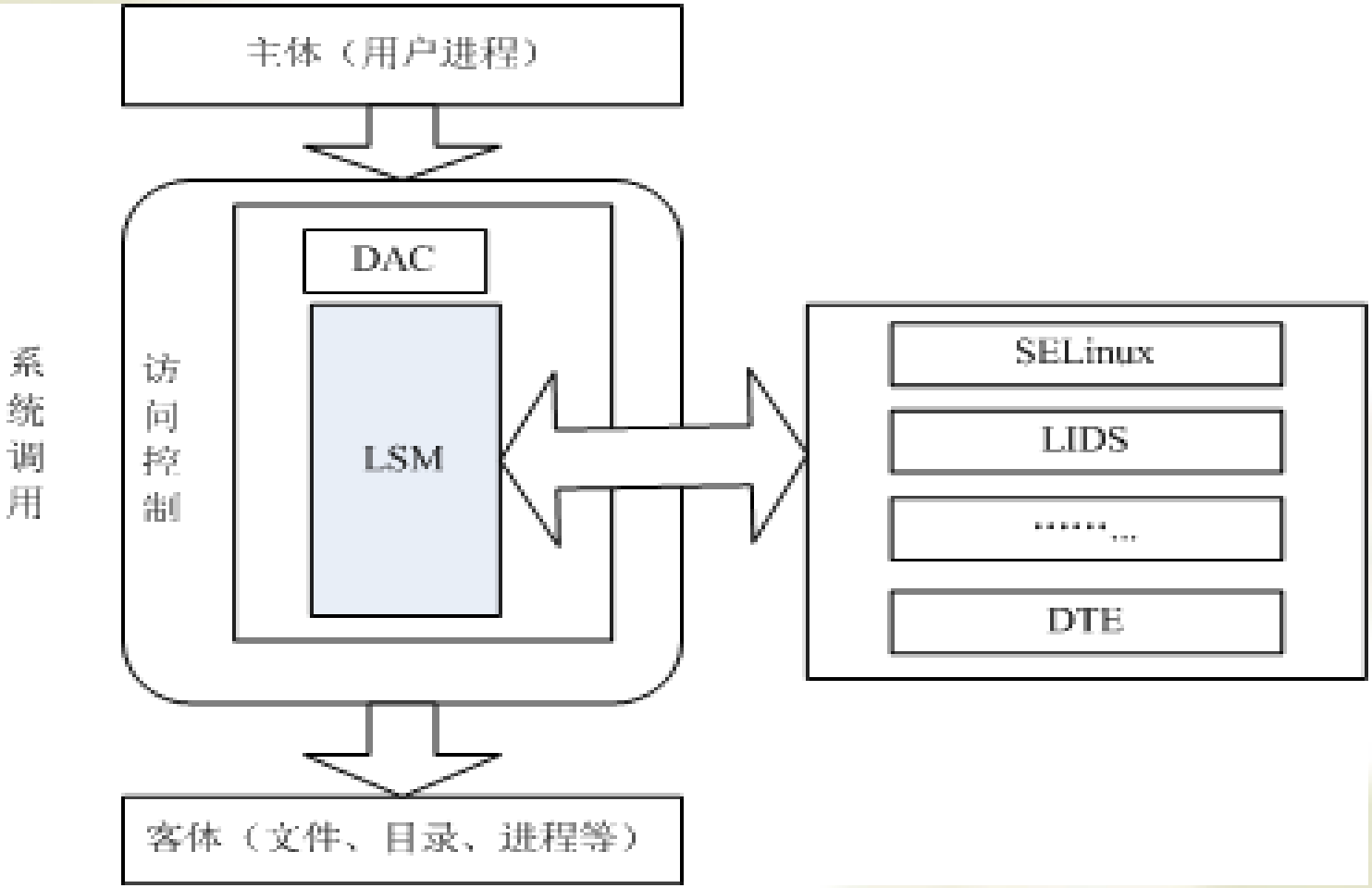
```
struct tasklet_struct {
    struct tasklet_struct *next;
    unsigned long state;
    atomic_t count;
    void (*func) (unsigned long);
    unsigned long data;
};
```

Linux内核安全框架
-Linux Security Module(LSM)

LSM 设计原则和目标

- 通用性。当选择一个使用不同安全策略的模型时，仅仅是装载一个不同的内核模块，不再需要额外的内核补丁。
- 简单性。LSM的实现应不涉及复杂的安全理论，不干扰或不对安全模型的实现提供“具体的”支持；尽量减少对原有内核的改动；不能对内核执行效率有大的影响。
- 能够使已有的POSIX.1e capabllities逻辑结构作为一个可选择的安全模块装入。

LSM框架



LSM 设计原则和目标

- 在内核系统调用处理中，提供调用策略的接口
- 提供包含安全钩子函数的结构体，及其内核注册、卸载函数
- 在内核核心数据结构中，提供字段，保存对应策略对应数据
- 提供额外的安全系统调用，方便实现对应策略的内核与用户态交互

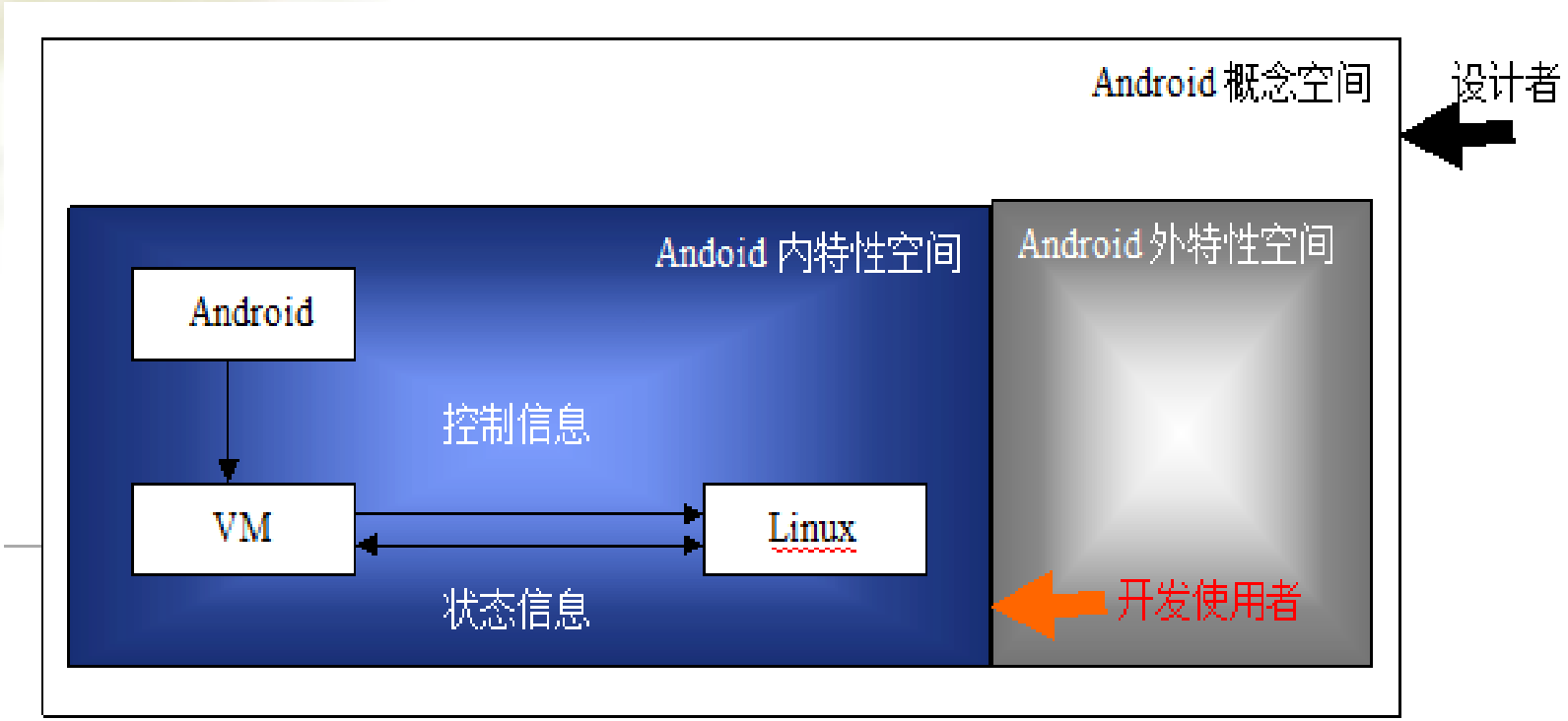
Linux内核中的其他机制

- 同步机制
- 审计机制
- Netfilter机制
- Inotify机制
- ……

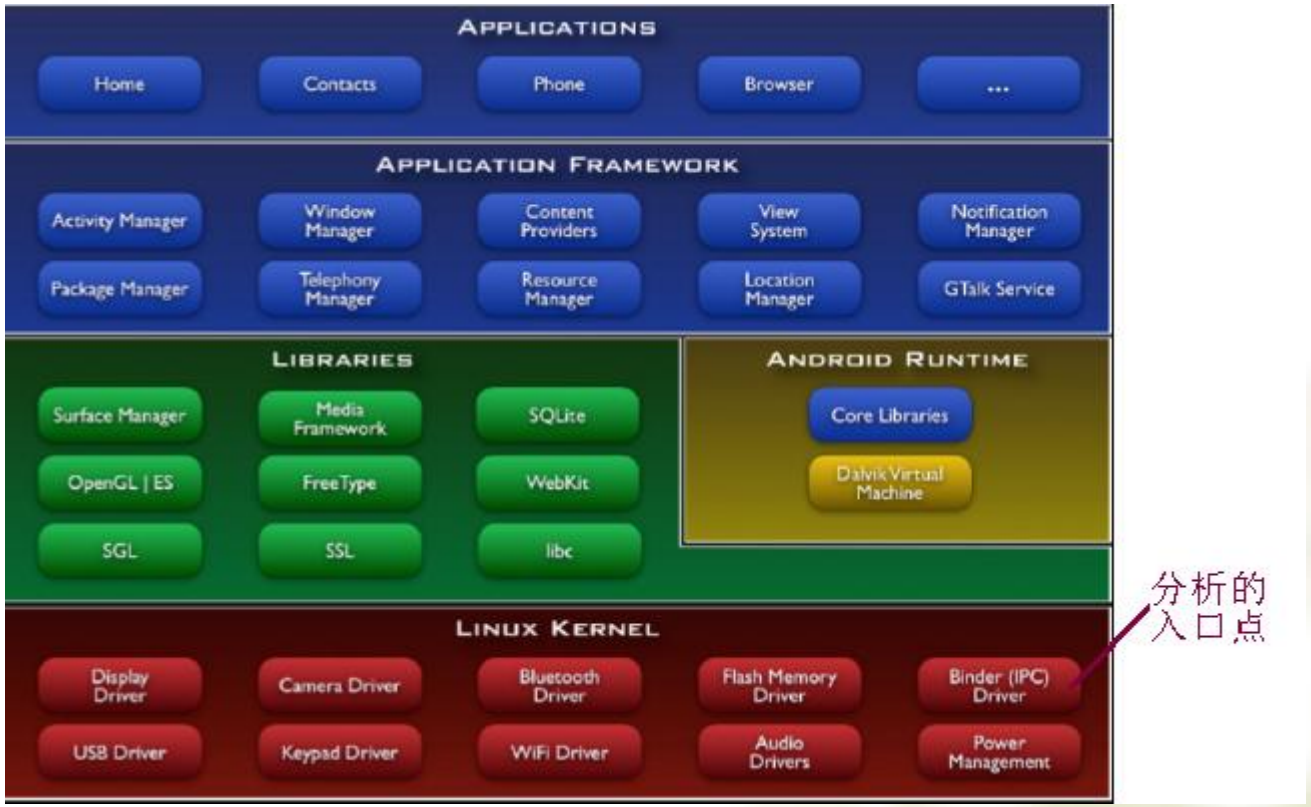
云时代终端操作系统
-Android

Android来了

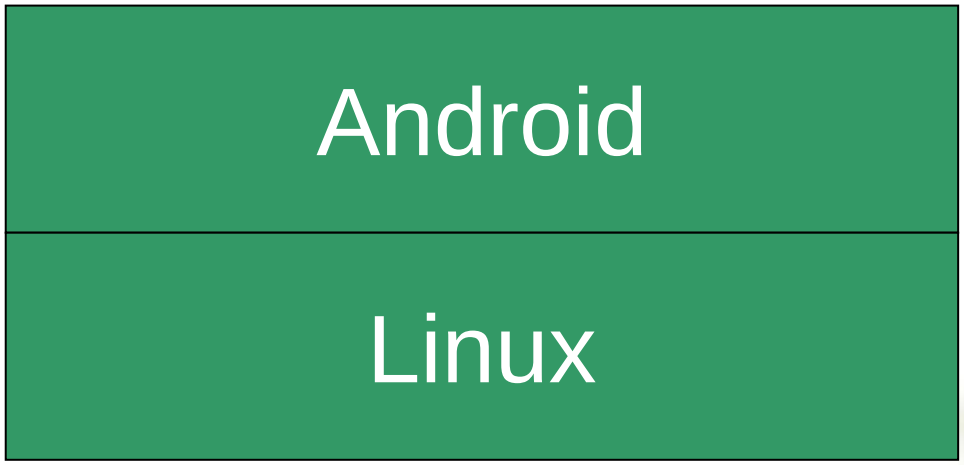
从设计意图出发


Android 概念空间
设计者
Andoid 内特性空间
Android 外特性空间
Android
控制信息
VM
Linux
状态信息
开发使用者

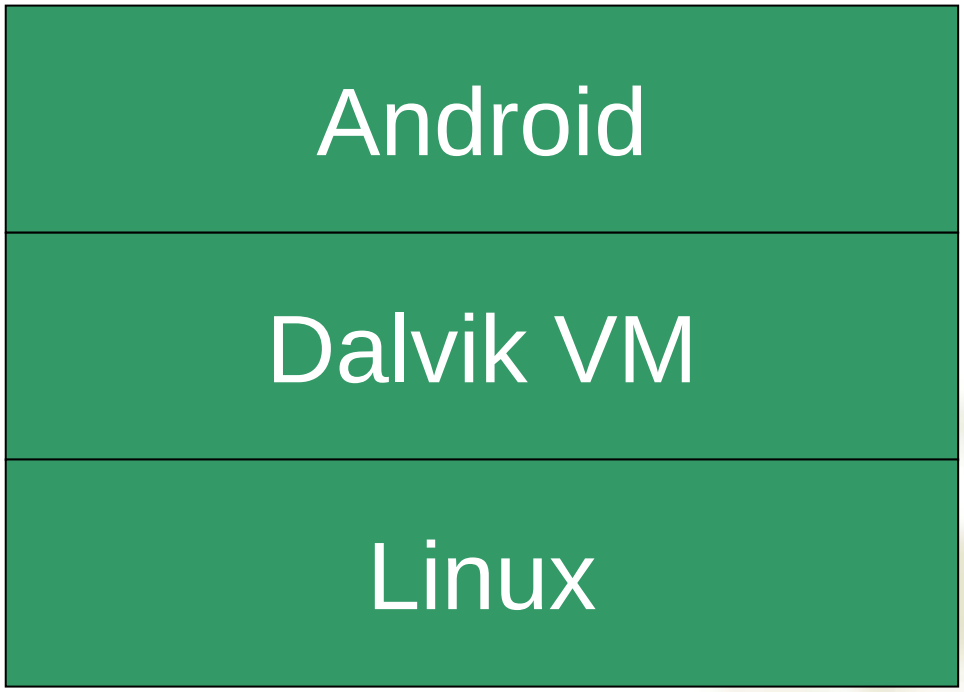
Android框架


APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
GTalk Service
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management
分析的入口点

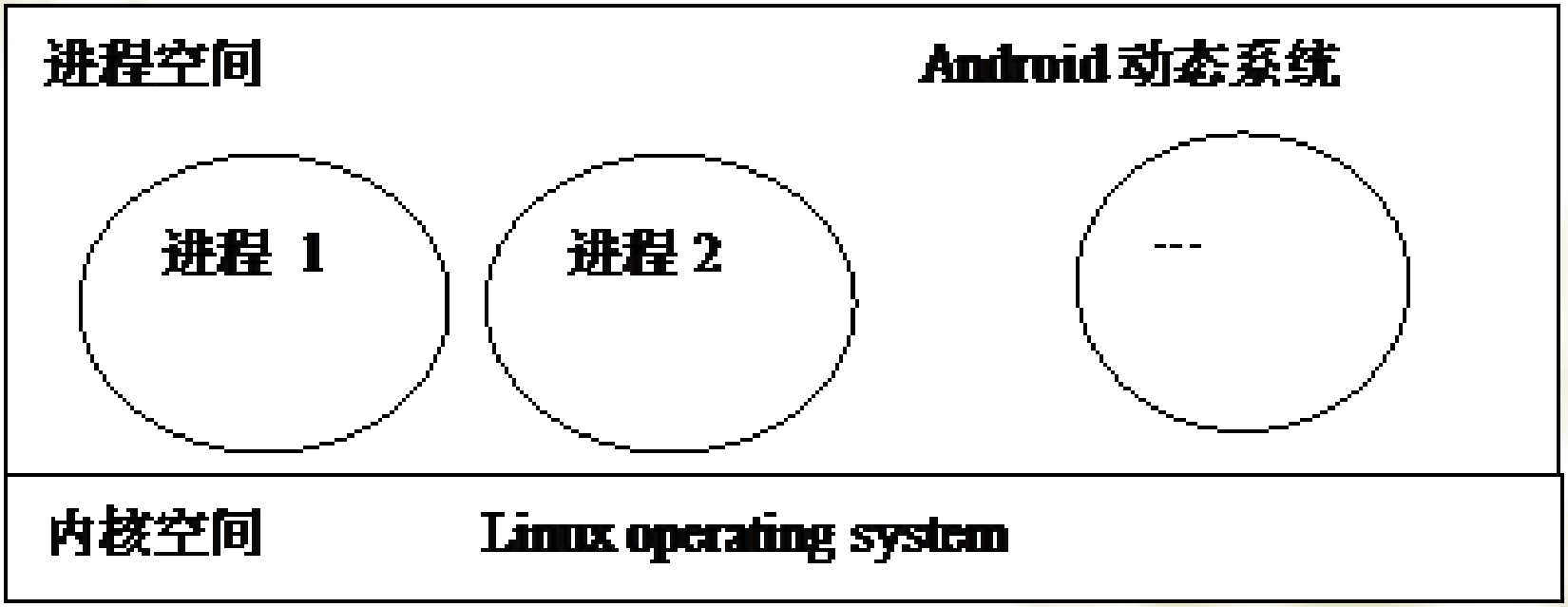
Android空间的演绎（一）



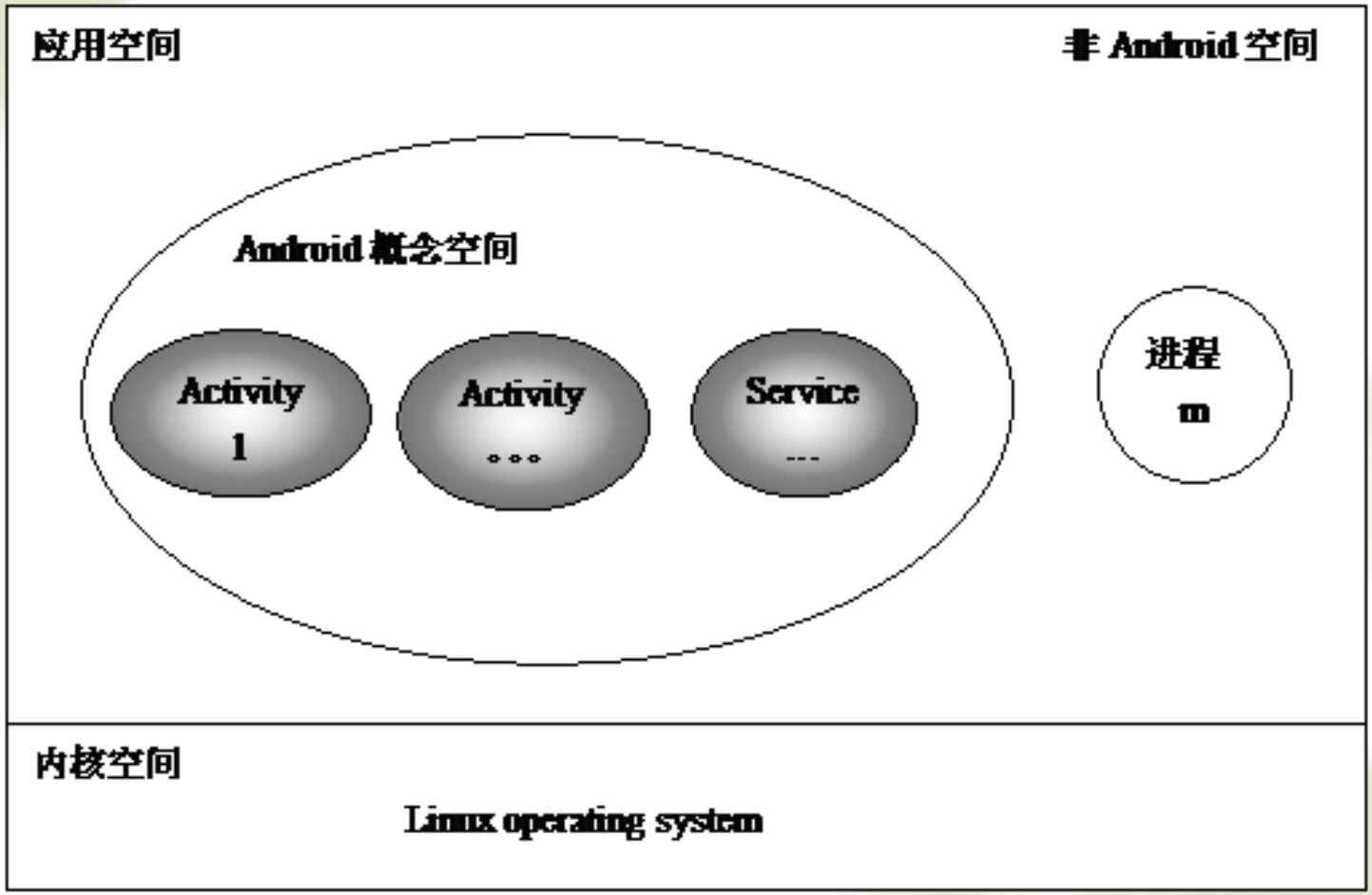
Android空间的演绎（二）



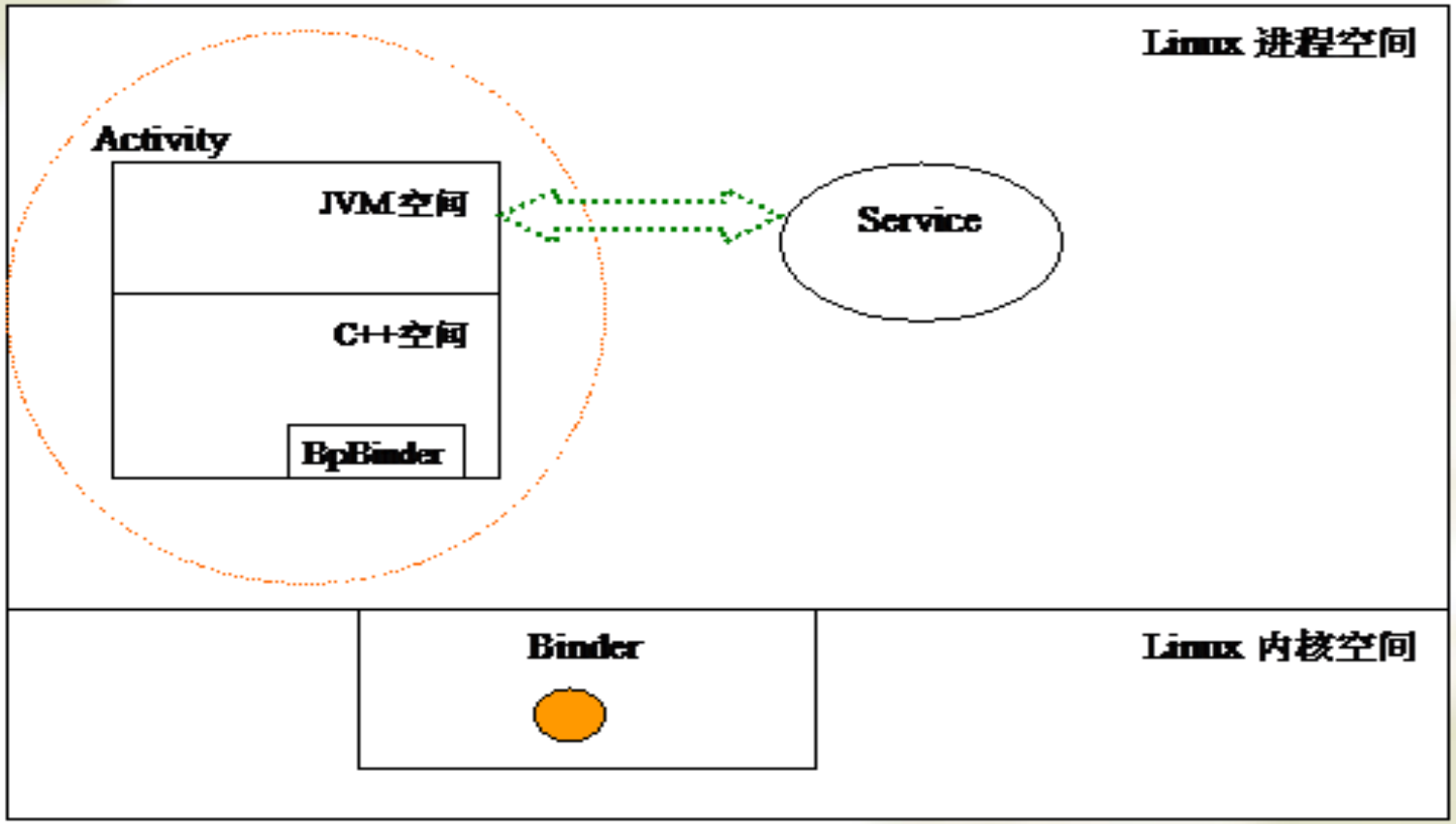
Android空间的演绎（三）



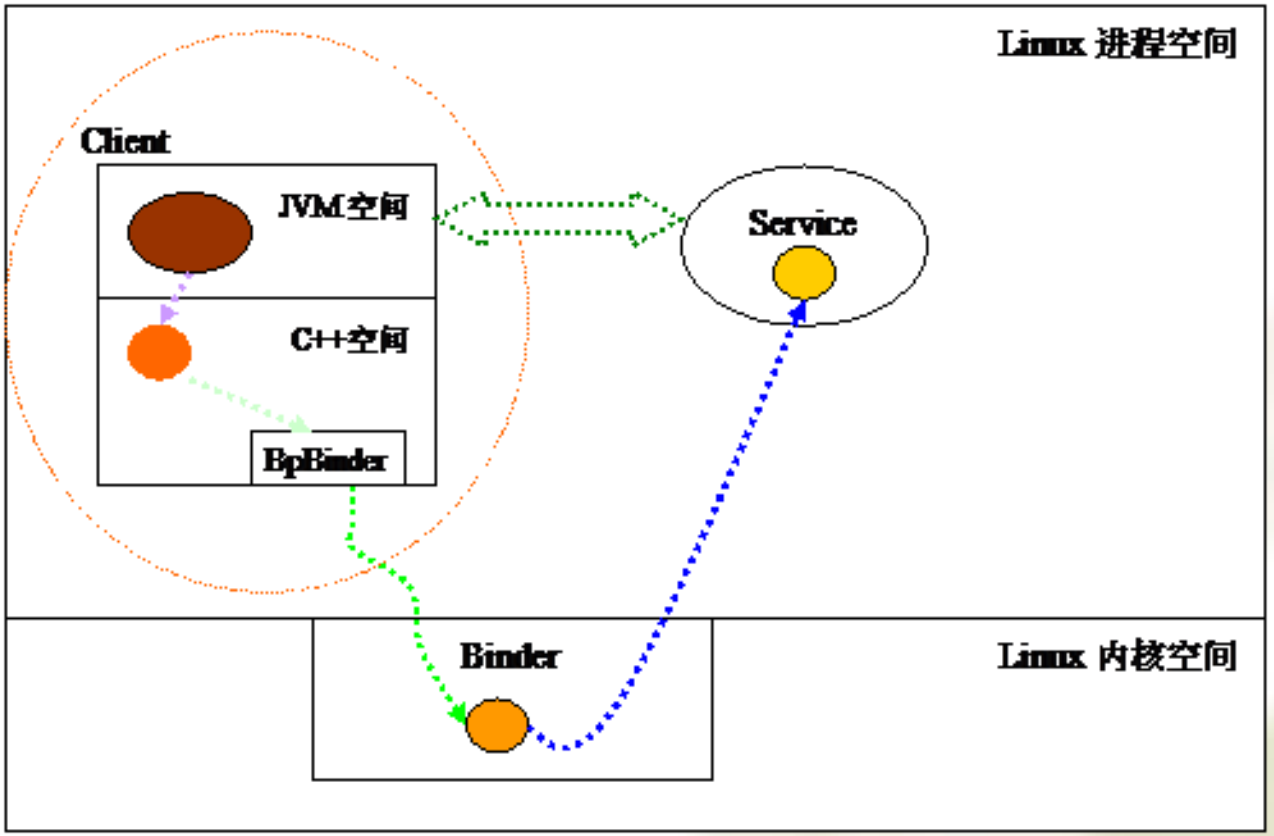
Android空间的演绎（四）



Android空间的演绎（五）



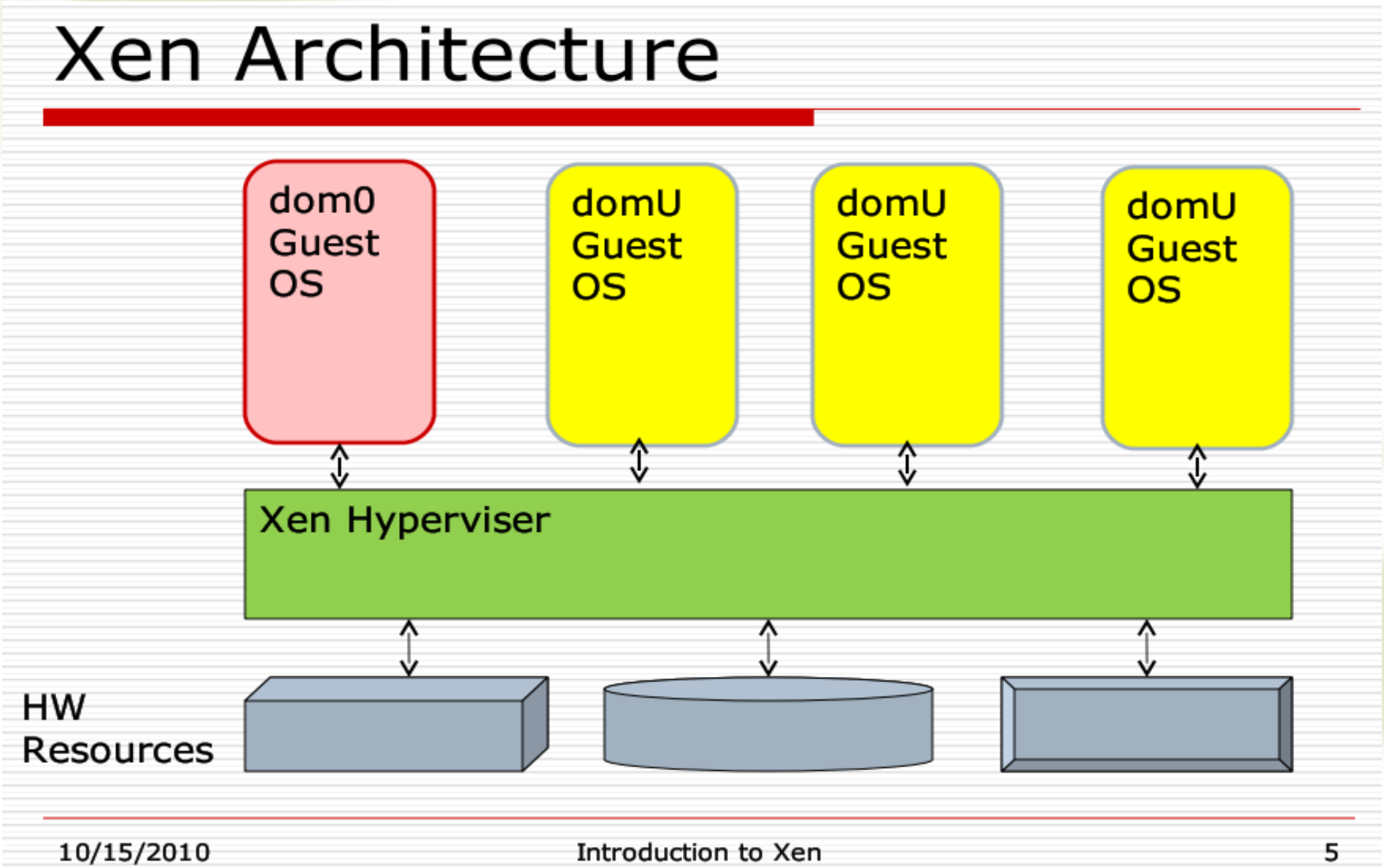
Android的核心机制binder



云计算核心技术

——虚拟化

xen体系结构


Xen Architecture
dom0
Guest
OS
domU
Guest
OS
domU
Guest
OS
domU
Guest
OS
Xen Hyperviser
HW
Resources
10/15/2010
Introduction to Xen
5

Xen Hypervisor

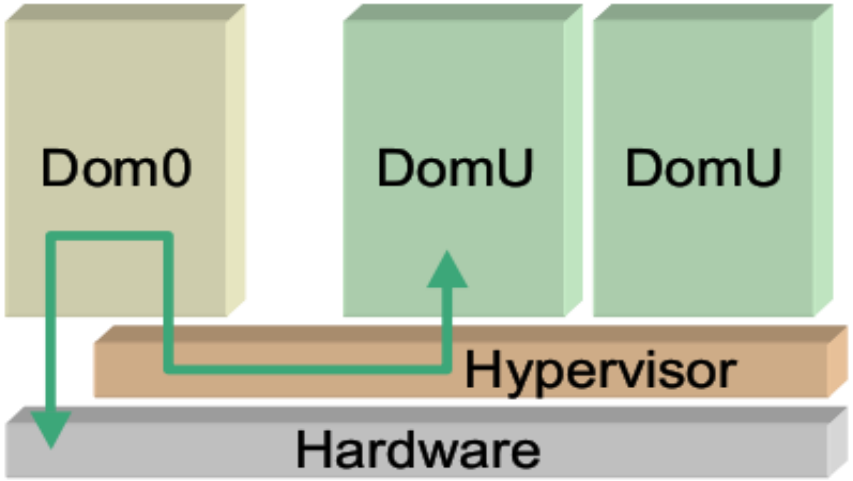
Xen Hypervisor

- Xen Hypervisor是一个介于硬件和操作系统之间的软件层，它负责在各虚拟机之间进行CPU调度和内存分配。Xen Hypervisor不仅抽象出硬件层，同时控制虚拟机的执行，因为这些虚拟机共享同一个处理环境。Xen Hypervisor不会处理网络、存储设备、视频以及其他I/O。

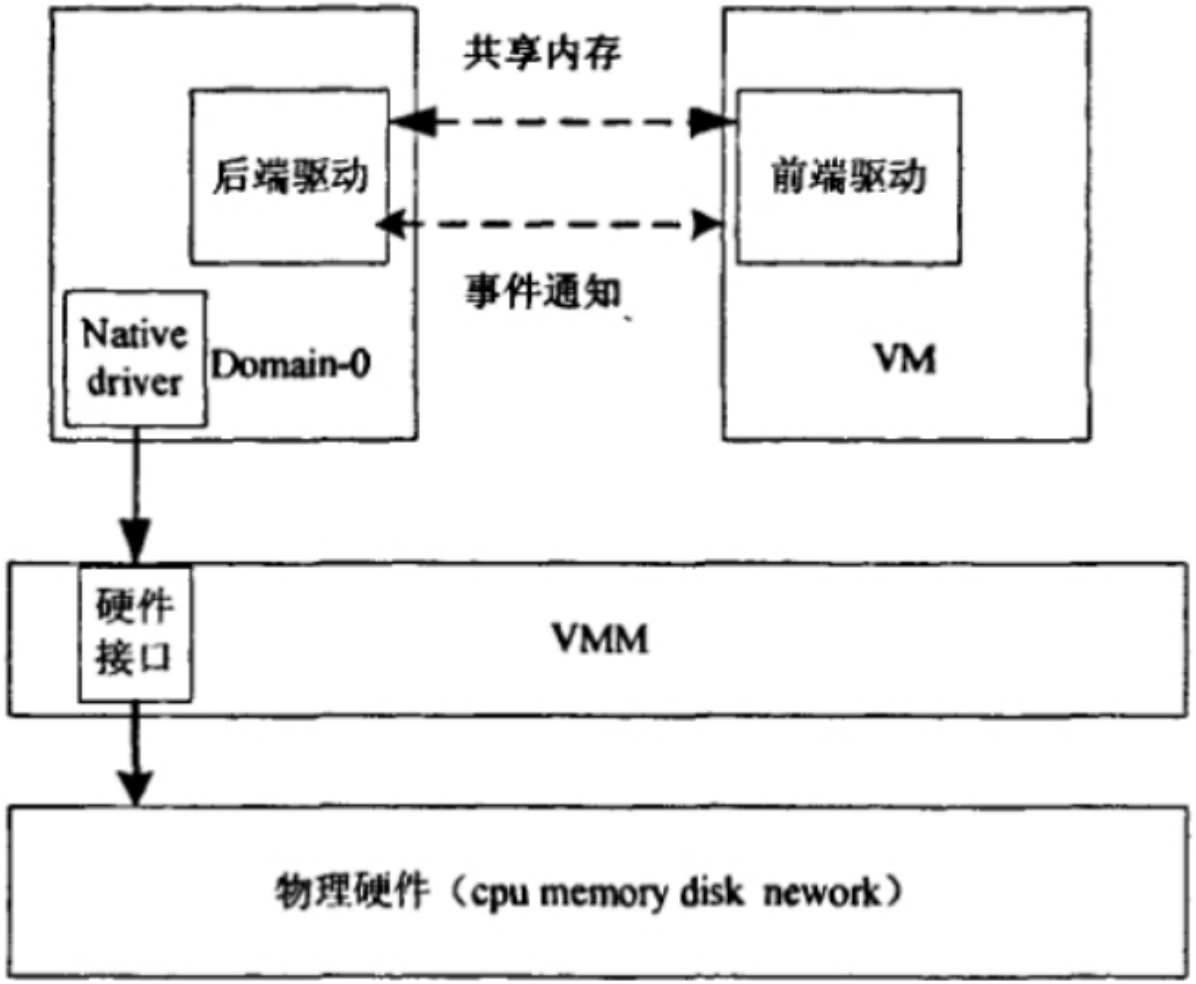
xen的驱动分离机制

I/O in Xen: the front-end and back-end

- All 'real' drivers live in Dom0
- DomU kernels have pseudo drivers that communicate with Dom0 via the hypervisor
- Necessary because only Dom0 is 'trusted'



虚拟设备模型


共享内存
后端驱动
前端驱动
事件通知
Native driver
Domain-0
VM
硬件接口
VMM
物理硬件（cpu memory disk nework）

机制与策略分离

- Xen hypervisor 实现了机制，而把策略留给domain 0客户操作系统。
- Xen 本身并不支持任何设备，它只是提供一种机制，使得客户操作系统可以直接访问物理设备。
- 客户操作系统可以使用现有的设备驱动程序。
- 这就使得即使支持一个新设备无需修改xen

进行着的开源项目

-弹性云存储

http://code.google.com/p/cloudxy/

背景介绍

- 弹性云平台是为用户提供按需分配物理资源的基础服务平台——也就是所谓的基础设施即服务（Infrastructure As a Service）。
- 平台主要借助虚拟机技术实现物理计算机资源（计算资源、存储资源、网络资源等）的再划分、隔离、以及现场分配和回收。
- 用户以完全独占方式获得虚拟机实例（其上已运行标准操作系统），具体使用方式也由用户自裁。

功能介绍

- 类似于EC2弹性云平台
- 按用户需求自动分配、管理、监控虚拟机
- 支持网络虚拟化
- 实现镜像存储的集中化--在hadoop上实现log structure filesystem(HLFS)作为虚拟机镜像存储后台
- 实现虚拟机动态调度
- 实现虚拟机服务和数据分析等离线服务混合部署

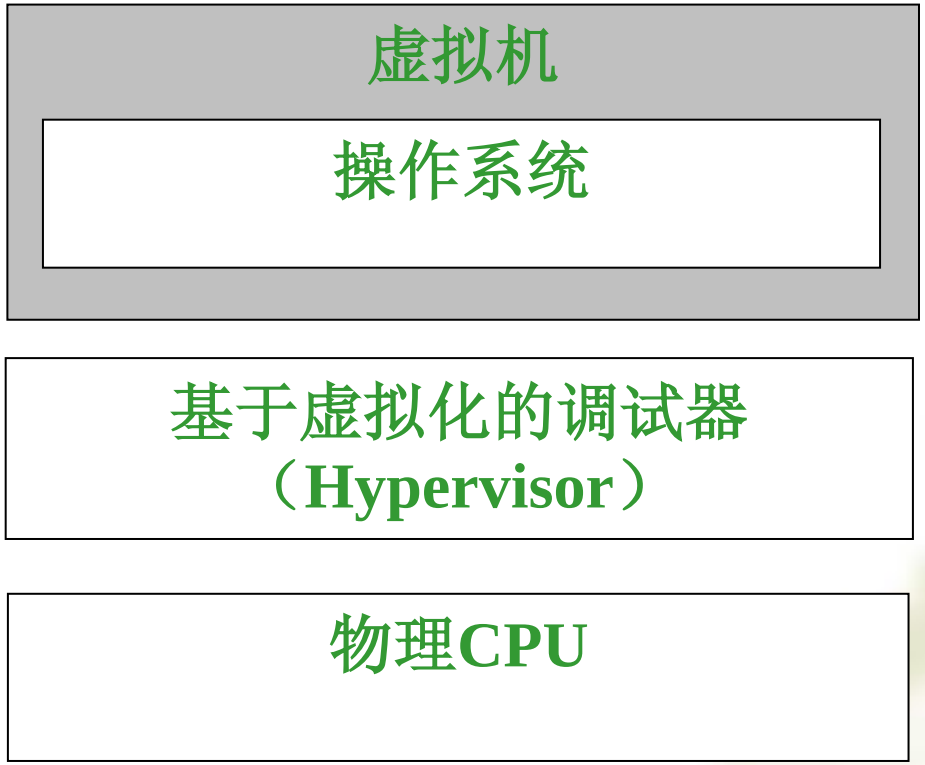
机制与策略分离

- Hdfs提供了分布式文件系统的良好机制
- HLFS建立在hdfs之上，具有高可用性、写透、快照、分布式、用户态

开源项目

-基于Intel VT技术的Linux内核调试器

基于Intel VT技术的Linux内核调试器



VMM的概念

在硬件虚拟化技术中，一个重要的概念就是VMM（Virtual Machine Monitor），虚拟机监控者，也就是Hypervisor，它是使用硬件虚拟化技术时创建的特权层，该层提供给虚拟机开发者，用来实现虚拟硬件与真实硬件的通信和一些事件处理操作。

Intel VT技术

VT技术的CPU支持2种运行模式

- VMX模式（启用虚拟化）
- Non-VMX模式（未启用虚拟化）

Intel VT技术

VMX模式中两种处理模式

- **VMX root模式**
- **　支持VT指令集，提供给VMM使用，可以对虚拟机进行管理**
- **VMX non-root模式**
- **　不支持VT指令集，是虚拟机运行的环境，被VMM控制**

VMM与虚拟机环境的切换

- VM Entry (VMX root >>> VMX non-root)
- 从VMM进入VM叫做VM Entry。例如创建一个虚拟机、VMM处理完虚拟机事物后返回虚拟机，都是VM Entry。
- VM Exit (VMX non-root >>> VMX root)
- 从VM退出返回到VMM叫VM Exit。例如VM访问硬件设备、CPU异常、外部中断、内存换页、VMCall等将触发VM Exit。

传统内核调试器工作原理

- 接管INT1、INT3异常处理程序（Hook中断处理函数）
- 设置断点
- 断点触发CPU异常，进入调试器
- 调试器决定异常是否传递给操作系统

如何将虚拟化技术应用在调试器

- 进入VMX root模式
- 设置虚拟机环境（VMCS），将当前CPU上下文完整复制到虚拟机环境。
- 启动虚拟机
- 设置断点
- 断点触发CPU异常，引发VM Exit，VMM处理CPU异常。
- 调试器决定异常是否传递给操作系统

调试器为什么位于虚拟化层

将虚拟化技术应用在调试器，可以完全避免被调试程序检测到当前自己正在被调试。

关键在于将现在运行的操作系统放置在虚拟机中。CS DS ES...EIP ESP...

本质：调试器作为VMM，优先响应CPU异常。

VMM作为机制

- 软件加密
- Rootkit（系统级木马）
- 调试器

无处不在的机制与策略分离...

3Q