

操作系统 Operation System

第二章 系统引导

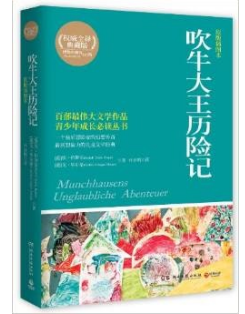
内容提要

- 计算机的启动过程 (X86)
- X86下Linux系统引导过程

Bootstrapping

- Bootstrap: 拔靴带 / 引导程序

- 吹牛大王历险记: **pull oneself up by one's bootstraps**
- Baron Munchausen pulls himself and his horse out of a swamp by his hair.



underwater



riding the cannon ball



flying with ducks

We are not alone

- 早期汽车的启动过程：
 - 需要对压缩的燃料和空气点火后引擎才能工作
 - 只有引擎工作了才能压缩燃料空气
 - 矛盾？
- 现在汽车的启动过程
 - 借助外力：人力、电机



启动，是一个很“纠结”的过程

- 现代计算机 —— 硬件 + 软件
- 启动的矛盾：
 - 一方面：必须通过程序控制使得计算机硬件进入特定工作状态（硬件启动依赖软件）
 - 另一方面：程序必须运行在设置好工作模式的硬件环境上（软件运行依赖硬件）
- 因此：启动前硬件状态必须假设在一个最安全、通用，因此也是功能最弱的状态，需要逐步设置硬件，以提升硬件环境能力
- OS启动是一个逐步释放系统灵活性的过程

X86 启动过程（与OS无关）

1. Turn on
2. CPU jump to physical address of BIOS (0xFFFFF0) (Intel 80386)
3. BIOS runs POST (Power-On Self Test)
4. Find bootable devices
5. Loads boot sector from MBR
6. BIOS yields control to OS BootLoader

第一阶段

第二阶段

BIOS (Basic Input/Output System)

- BIOS设置程序是被固化到电脑主板上的ROM芯片中的一组程序，其主要功能是为电脑提供最底层地、最直接地硬件设置和控制。BIOS通常与硬件系统集成在一起（在计算机主板的ROM或EEPROM中），所以也被称为**固件**。



BIOS on board



BIOS on screen

BIOS

- BIOS程序存放于一个断电后内容不会丢失的只读存储器中；系统过电或被重置（reset）时，处理器要执行第一条指令的地址会被定位到BIOS的存储器中，让初始化程序开始运行。
- 在X86系统中，CPU加电后将跳转到BIOS的固定物理地址0xFFFF0。

(Intel 80386)

启动第一步——加载BIOS

- 当打开计算机电源，计算机会首先加载BIOS信息。BIOS中包含了CPU的相关信息、设备启动顺序信息、硬盘信息、内存信息、时钟信息、PnP特性等等。在此之后，计算机心里就有谱了，知道应该去读取哪个硬件设备了。

BIOS

硬件自检 (Power-On Self-Test)

- BIOS代码包含诊断功能，以保证某些重要硬件组件，像是键盘、磁盘设备、输出输入端口等等，可以正常运作且正确地初始化。

如果硬件出现问题，主板会发出不同含义的蜂鸣，启动中止。如果没有问题，屏幕就会显示出CPU、内存、硬盘等信息。

Diskette Drive B : None									
Serial Port(s) : 3F0 2F0									
Pri. Master Disk : LBA,ATA 100, 250GB									
Parallel Port(s) : 370									
Pri. Slave Disk : LBA,ATA 100, 250GB									
DDR at Bank(s) : 0 1 2									
Sec. Master Disk : None									
Sec. Slave Disk : None									
Pri. Master Disk HDD S.M.A.R.T. capability ... Disabled									
Pri. Slave Disk HDD S.M.A.R.T. capability ... Disabled									
PCI Devices Listing ...									
Bus	Dev	Fun	Vendor	Device	SUID	SSID	Class	Device Class	IRQ
0	27	0	8086	2668	1458	A005	0403	Multimedia Device	5
0	29	0	8086	2658	1458	2658	0C03	USB 1.1 Host Cntrlr	9
0	29	1	8086	2659	1458	2659	0C03	USB 1.1 Host Cntrlr	11
0	29	2	8086	265A	1458	265A	0C03	USB 1.1 Host Cntrlr	11
0	29	3	8086	265B	1458	265A	0C03	USB 1.1 Host Cntrlr	5
0	29	7	8086	265C	1458	5006	0C03	USB 1.1 Host Cntrlr	9
0	31	2	8086	2651	1458	2651	0101	IDE Cntrlr	14
0	31	3	8086	266A	1458	266A	0C05	SMBus Cntrlr	11
1	0	0	10DE	0421	10DE	0479	0300	Display Cntrlr	5
2	0	0	1283	8212	0000	0000	0180	Mass Storage Cntrlr	10
2	5	0	11AB	4320	1458	E000	0200	Network Cntrlr	12
								ACPI Controller	9

BIOS

读取启动顺序 (Boot Sequence)

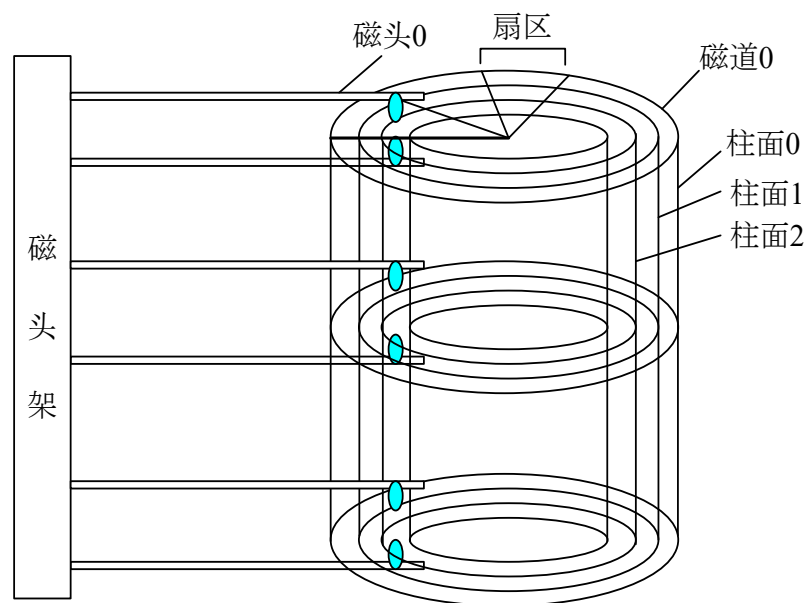
- 现代的BIOS可以让用户选择由哪个设备引导电脑，如光盘驱动器、硬盘、软盘、USB U盘等等。这项功能对于安装操作系统、以CD引导电脑、以及改变电脑找寻开机媒体的顺序特别有用。

打开BIOS的操作界面，里面有一项就是"设定启动顺序"。



磁盘的寻址

- 磁道 (track)
 - 盘片上以盘片中心为圆心，不同半径的同心圆。
- 扇区 (Sector)
 - 盘片被分成许多扇形的区域
- 柱面 (Cylinder)
 - 硬盘中，不同盘片相同半径的磁道所组成的圆柱。
- 每个磁盘有两个面，每个面都有一个磁头(Head)。
- 磁盘按照C-H-S规则寻址



BIOS的问题

- 16位~20位实模式寻址能力
- 实现结构、可移植性
 - 磁盘的Cylinder-Head-Sector (CHS) 寻址
 - 一个扇区512byte, MBR只有32位来存取扇区数
 - $2^{32} * 512 \text{ byte} = 2199023 \text{ MB} = 2.19 \text{ TB}$
- 问题根源
 - 历史的局限性、向前兼容的压力
 - 支持遗留软件: 老设备驱动等
 - 经典 \approx (成熟、稳定、共识), 来之不易, 维持整个产业生态正常运转的必要Tradeoff
 - IT发展太快, 对“历史局限”的继承, 导致改变成本越来越高。——“另起炉灶”(UEFI) 来解决。

UEFI——统一可扩展固件接口



- Unified Extensible Firmware Interface
 - 2000年提出，Intel组建生态
- 功能特性
 - 支持从超过2TB的大容量硬盘引导 (GUID Partition Table, GPT分区) (硬件支持)
 - CPU-independent architecture(可移植性)
 - CPU-independent drivers (可移植性)
 - Flexible pre-OS environment, including network capability (硬件支持)
 - Modular design (可移植性)

https://en.wikipedia.org/wiki/Unified_Extensible_Firmware_Interface#cite_note-note1-15

UEFI和BIOS的比较

二者显著的区别是：

- EFI是用模块化，C语言风格的参数堆栈传递方式，动态链接的形式构建的系统，较BIOS而言更易于实现，容错和纠错特性更强，缩短了系统研发的时间。
- 它运行于32位或64位模式，乃至未来增强的处理器模式下，突破传统BIOS的16位代码的寻址能力，达到处理器的最大寻址。

启动第二步——读取MBR

- 硬盘上第0磁头第0磁道第一个扇区被称为MBR，也就是Master Boot Record，即主引导记录，它的大小是512字节，别看地方不大，可里面却存放了预启动信息、分区表信息。

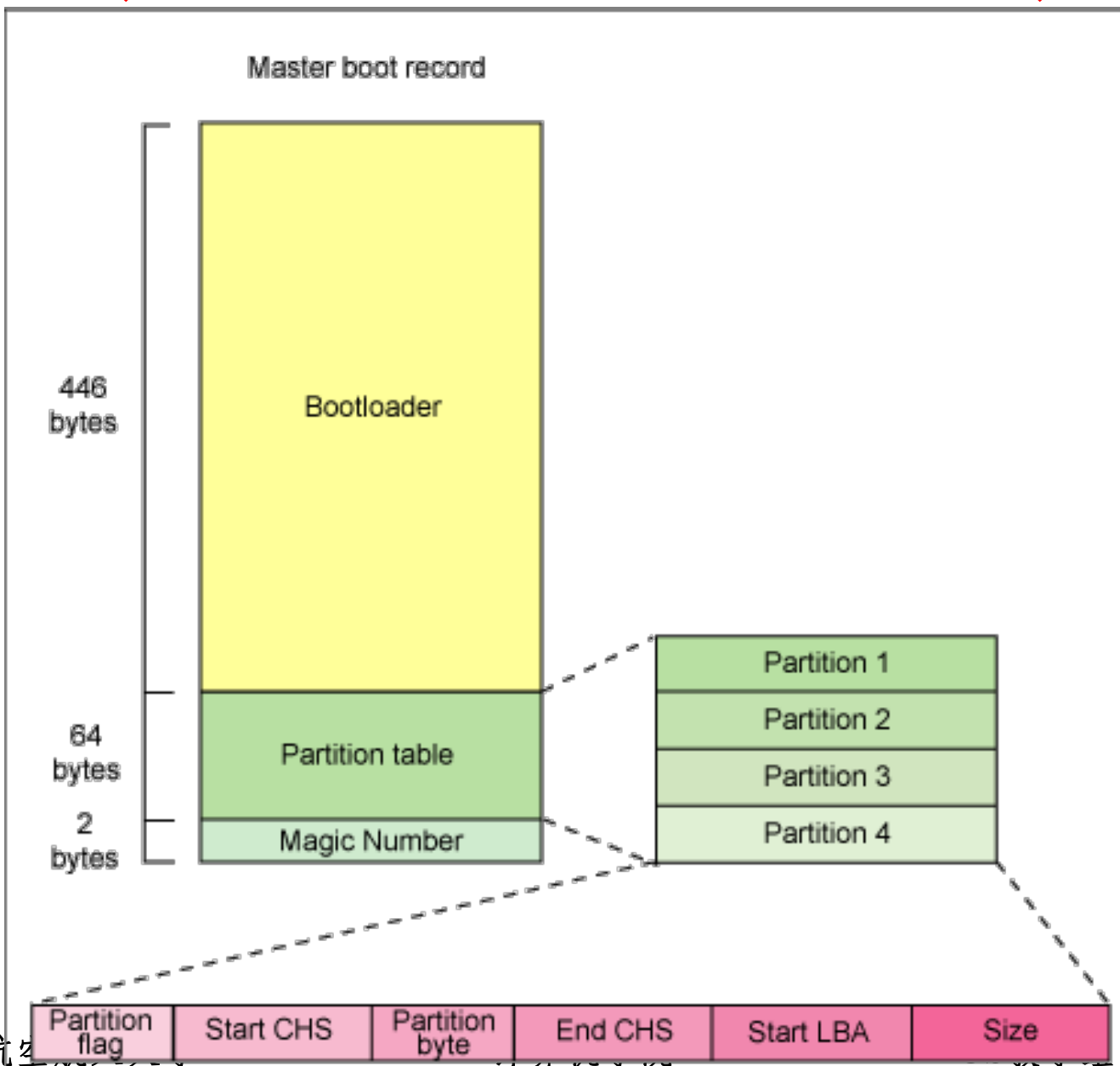
装入MBR

- MBR的全称是Master Boot Record（主引导记录），MBR早在1983年IBM PC DOS 2.0中就已经提出。之所以叫“主引导记录”，是因为它是存在于驱动器开始部分的一个特殊的启动扇区。
- 这个扇区包含了已安装的操作系统的启动加载器(BootLoader)和驱动器的逻辑分区信息。

MBR的结构

- MBR(Master Boot Record)主引导记录包含两部分的内容，前446字节为启动代码及数据；
- 之后则是分区表（DPT），分区表由四个分区项组成，每个分区项数据为16字节，记录了启动时需要的分区参数。这64个字节分布在MBR的第447-510字节。
- 后面紧接着两个字节AA和55被称为魔数(Magic Number), BOIS读取MBR的时候总是检查最后是不是有这两个magic number ,如果没有就被认为是一个没有被分区的硬盘。

MBR (Master Boot Record)



Partition flag	Start CHS	Partition byte	End CHS	Start LBA	Size
-------------------	-----------	-------------------	---------	-----------	------

存储字节位	内容及含义
第1字节	引导标志。若值为80H表示活动分区，若值为00H表示非活动分区。
第2、3、4字节	本分区的起始磁头号、扇区号、柱面号。其中： 磁头号——第2字节；扇区号——第3字节的低6位； 柱面号——为第3字节高2位+第4字节8位。
第5字节	分区类型符。 00H——表示该分区未用（即没有指定）； 06H——FAT16基本分区； 0BH——FAT32基本分区； 05H——扩展分区； 07H——NTFS分区； 0FH——（LBA模式）扩展分区（83H为Linux分区等）。
第6、7、8字节	本分区的结束磁头号、扇区号、柱面号。其中： 磁头号——第6字节；扇区号——第7字节的低6位； 柱面号——第7字节的高2位+第8字节。
第9-12字节	本分区之前已用了的扇区数。
第13,14,15,16字节	本分区的总扇区数。

MBR

- 由于MBR的限制 只能有4个主分区，系统必须装在主分区上面。
- 硬盘分区有三种，主磁盘分区、扩展磁盘分区、逻辑分区。
- 一个硬盘主分区至少有1个，最多4个，扩展分区可以没有，最多1个。且主分区+扩展分区总共不能超过4个。逻辑分区可以有若干个。
- 主分区只能有一个是活动的（active），其余为inactive。

MBR

- 分出主分区后，其余的部分可以分成扩展分区，一般是剩下的部分全部分成扩展分区。
- 扩展分区不能直接使用，必须分成若干逻辑分区。所有的逻辑分区都是扩展分区的一部分。

硬盘的容量 = 主分区的容量 + 扩展分区的容量

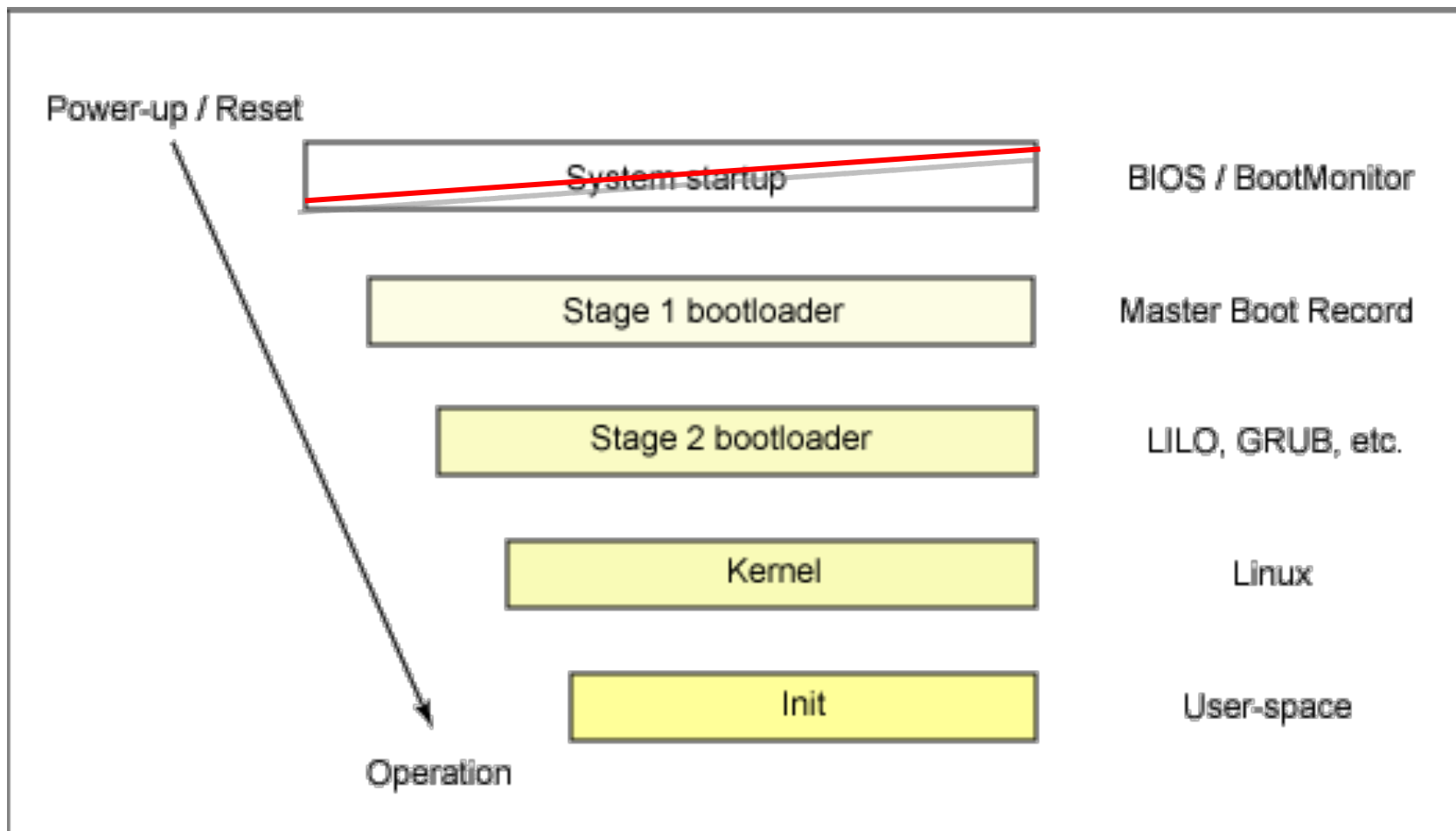
扩展分区的容量 = 各个逻辑分区的容量之和

内容提要

- 计算机的启动过程 (X86)
- X86下Linux系统引导过程

How Linux boot?

- 逐级引导、逐步释放灵活性



启动第三步——Boot Loader

- Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核做好一切准备。

Boot loader

- Boot loader 也可以称之为操作系统内核加载器(OS kernel loader), 通常是严重地依赖于硬件而实现的。
- GRUB 和 LILO 最重要的Linux加载器, 存放于MBR的bootloader区域。
 - Linux Loader (LILO)
 - GRand Unified Bootloader (GRUB)

LILLO: Linux LOader

- *Linux LOader (LILLO)* 已经成为所有 Linux 发行版的标准组成部分,是最老的 Linux 引导加载程序。
- 优点: 它可以快速启动安装在主引导记录中的 Linux 操作系统。
- 局限: LILLO 配置文件被反复更改时,主引导记录 (MBR) 也需要反复重写,但重写可能发生错误,这将导致系统无法引导。

GNU GRUB

- GNU GRand Unified Bootloader
 - 允许用户可以在计算机内同时拥有多个操作系统，并在计算机启动时选择希望运行的操作系统。

```
GNU GRUB  version 0.97  (637K lower / 1046400K upper memory)

Red Hat Enterprise Linux (2.6.32-279.el6.x86_64)
Windows XP SP3

Use the ↑ and ↓ keys to select which entry is highlighted.
Press enter to boot the selected OS, 'e' to edit the
commands before booting, 'a' to modify the kernel arguments
before booting, or 'c' for a command-line.
```

GRUB 与 LILO 的比较

- LILO 没有交互式命令界面，而 GRUB 拥有。
- LILO 不支持网络引导，而 GRUB 支持。
- LILO 存储在 MBR 中。如果修改了 LILO 配置文件会重写到 MBR。但错误配置的 MBR 可能会让系统无法引导。
- 使用 GRUB，如果配置文件配置错误，则只是默认转到 GRUB 命令行界面。

GRUB磁盘引导过程

- **stage1:** 读取磁盘第一个512字节（硬盘的0道0面1扇区，被称为MBR（主引导记录），也称为bootsect。执行主引导程序，进入Stage 1.5。
- **stage1.5:** 识别各种不同的文件系统格式。这使得grub识别到文件系统（OS文件系统还没就绪）
- **stage2:** 加载系统引导菜单(/boot/grub/menu.lst或grub.lst)，根据用户选择，加载内核映像(kernel image)到磁盘。
- 至此GRUB完成使命，内核开始接管。

运行主引导程序

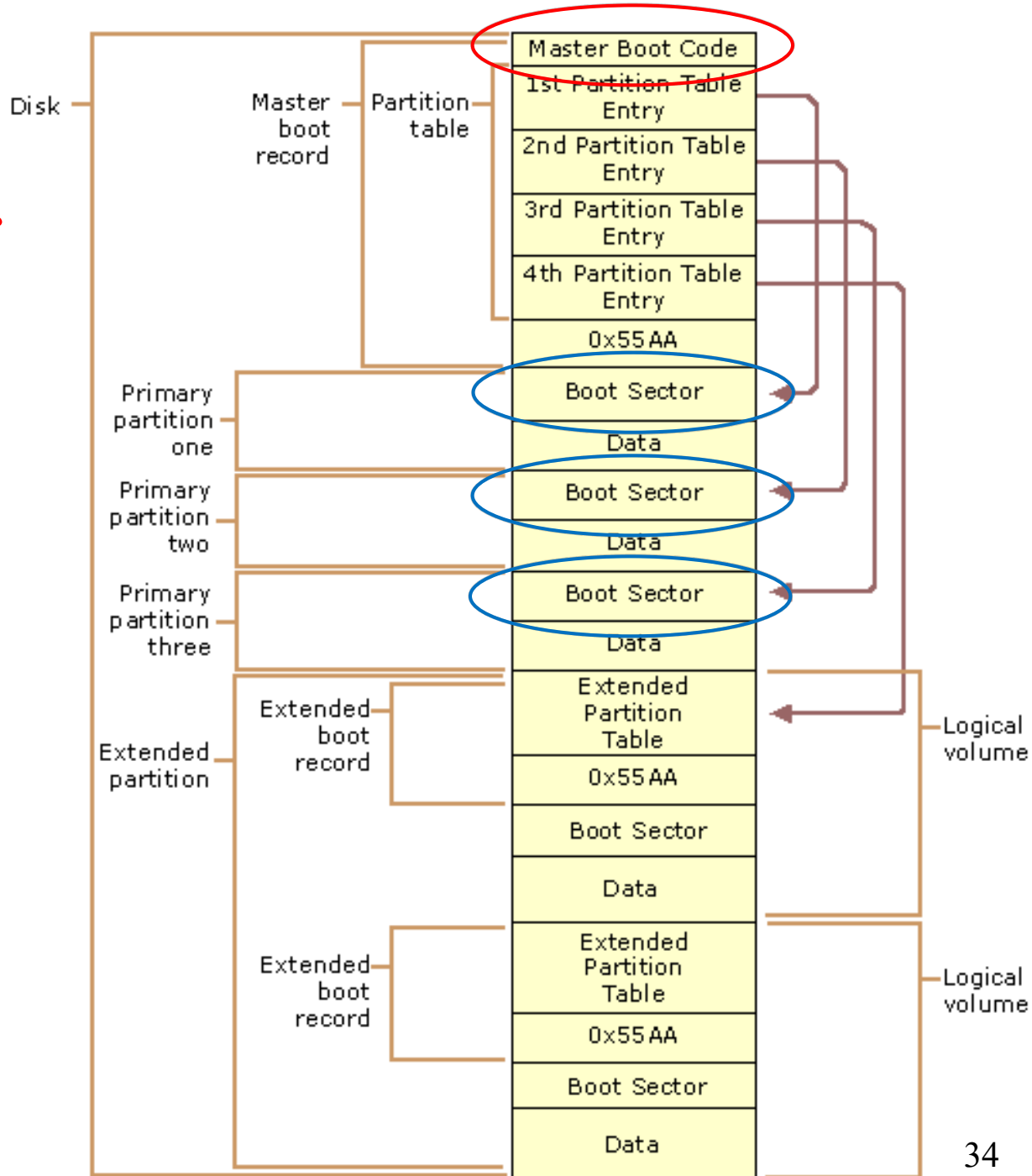
- BIOS将硬盘的主引导记录（位于0柱面、0磁道、1扇区）读入7C00处，然后将控制权交给主引导程序（GRUB的第一阶段）。任务包括：
 1. 检查是否有启动介质 “（WORD）0x7dfe是否等于0xaa55”。若不等于则转去尝试其他介质；如果没有其他启动介质，则显示 “No ROM BASIC”，然后死机；
 2. 跳转到0x7c00处执行引导程序；
 3. 将自己复制到0x0600处，然后继续执行；

运行主引导程序

4. 在主分区表中搜索标志为活动的分区。如果发现没有活动分区或者不止一个活动分区，则停止；
5. 将活动分区的第一个扇区读入内存地址0x7c00处；
6. 检查位于地址0x7dfe的“（WORD内容）是否等于0xaa55”，若不等于则显示“Missing Operating System”，然后停止，或尝试软盘启动；
7. 跳转到0x7c00处继续执行特定系统的启动程序；

MBR与引导扇区Boot Sector的关系

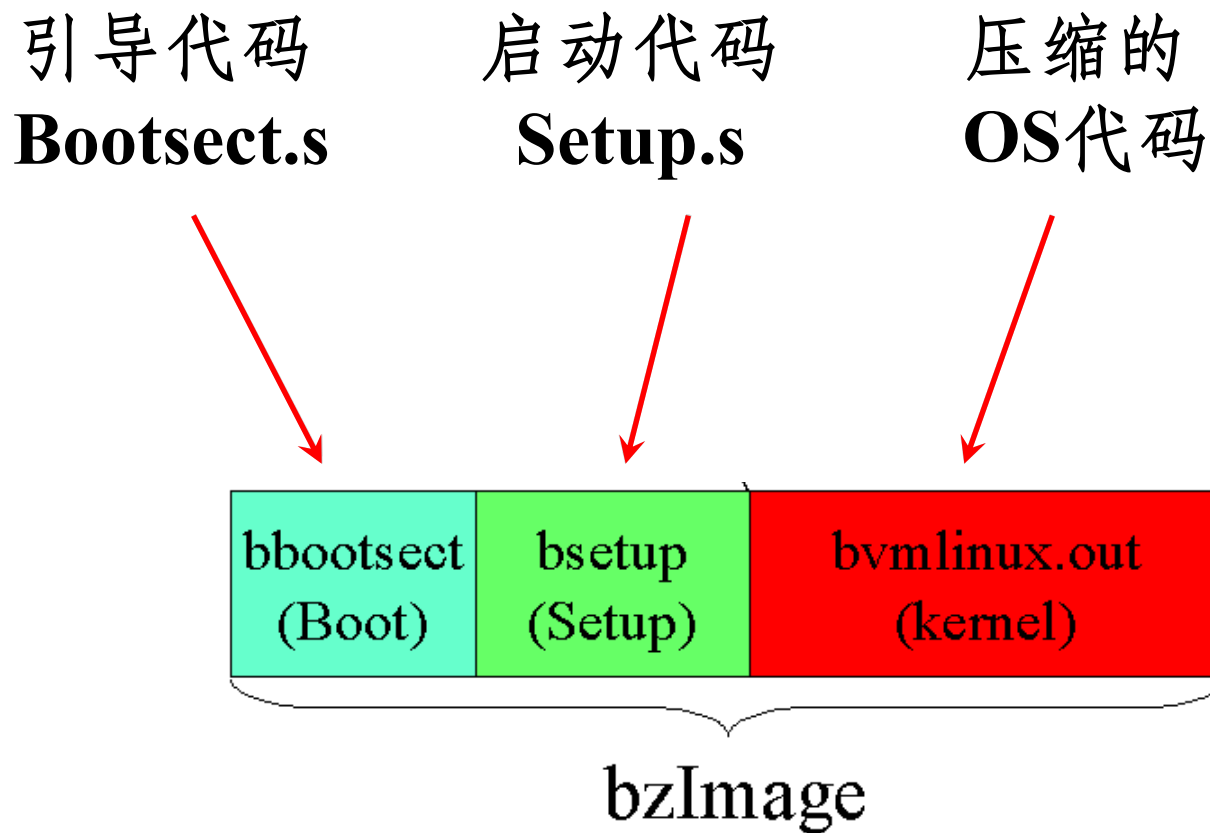
- **MBR**存放的位置是整个硬盘第一个扇区。
- **Boot Sector**是硬盘上每个分区的第一个扇区。



内核镜像

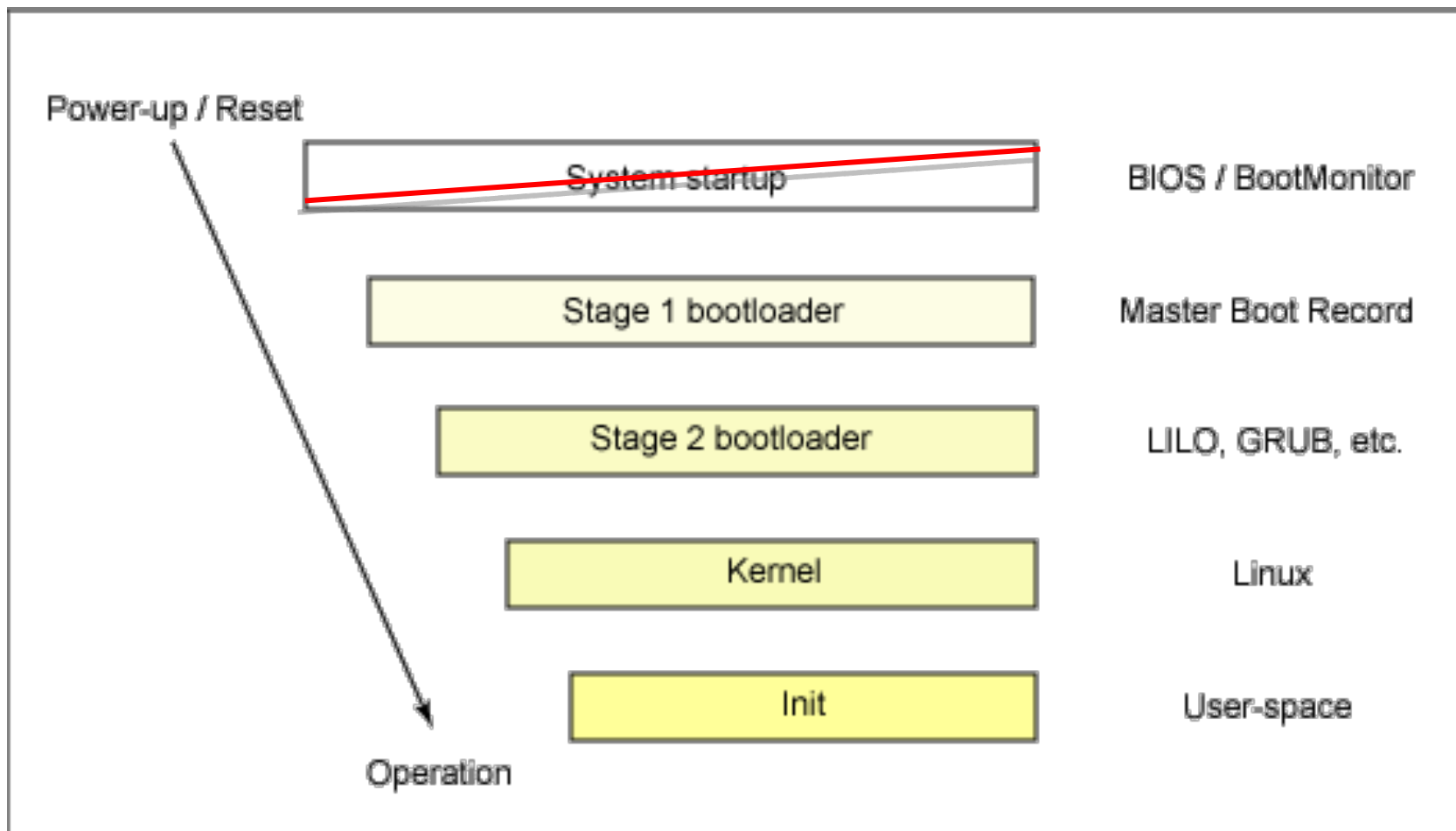
- 操作系统内核常驻内存直到电脑关闭
- 内核镜像并不是一个可执行的内核，而是内核的压缩后的镜像
 - `zImage` < 512 KB
 - `bzImage` > 512 KB

Kernel Image



How Linux boot?

- 逐级引导、逐步释放灵活性



启动第四步——加载内核

- 根据grub设定的内核映像所在路径，系统读取内核映像，并进行解压缩操作。
- 系统将解压后的内核放置在内存之中，初始化函数并初始化各种设备，完成Linux核心环境的建立。

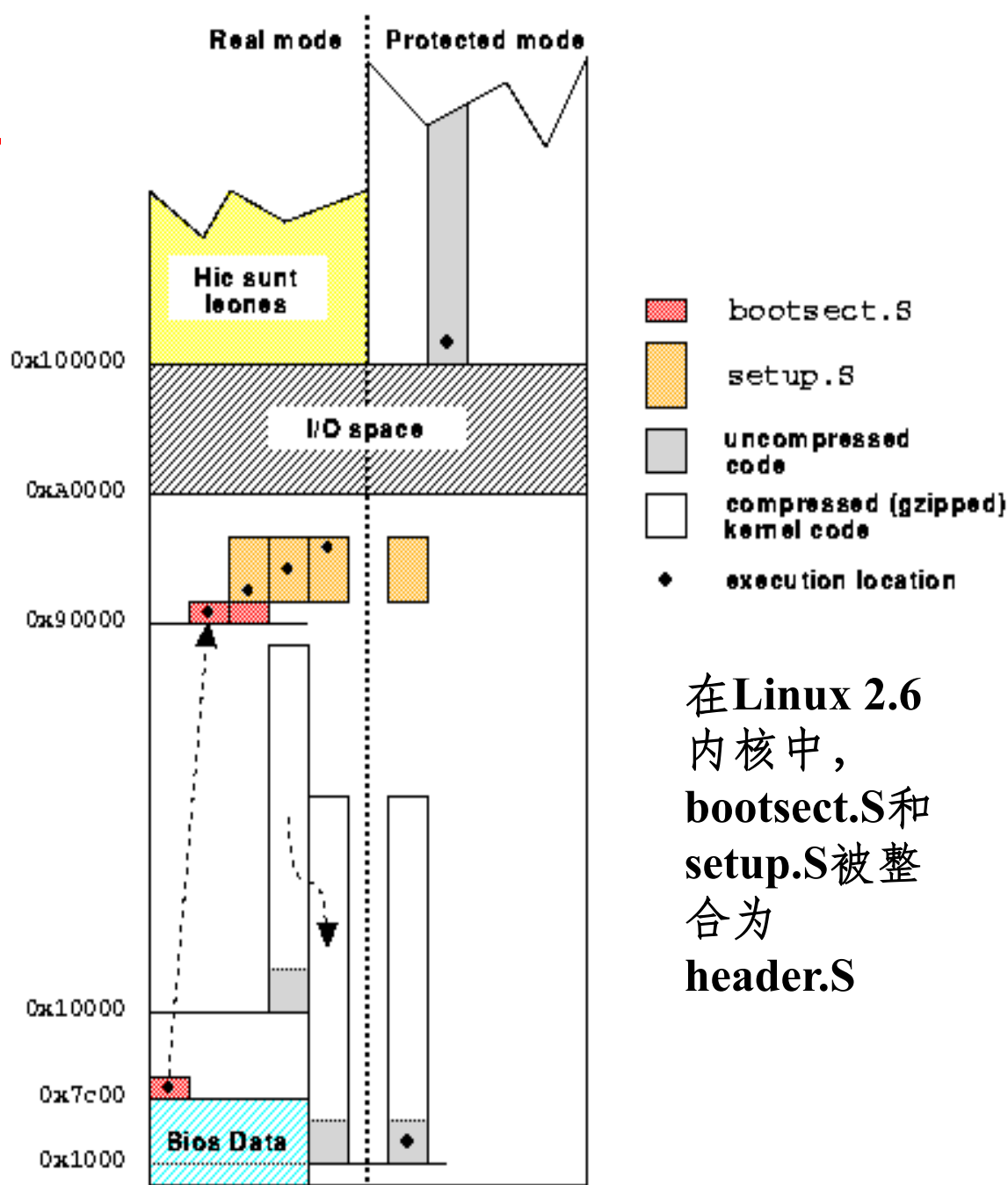
实模式和保护模式(X86)

- 实模式: 物理地址寻址
 - 8086 数据总线16位, 地址总线20位
 - 一次最多取数据 $2^{16}=64K$
 - 最大寻址 $2^{20}=1M$ 系统内存
 - 只能支持单任务OS
- 保护模式: 逻辑地址
 - 80286开始, 可以寻址全部内存
 - 进程运行在独自受保护的地址空间, 相互隔离
 - 支持多道程序设计 (多任务系统)
 - 支持虚拟存储
- 为了向前兼容保留实模式
 - 80386 (32位), 奔腾, 酷睿CPU都兼容实模式

Linux Kernel 加载过程

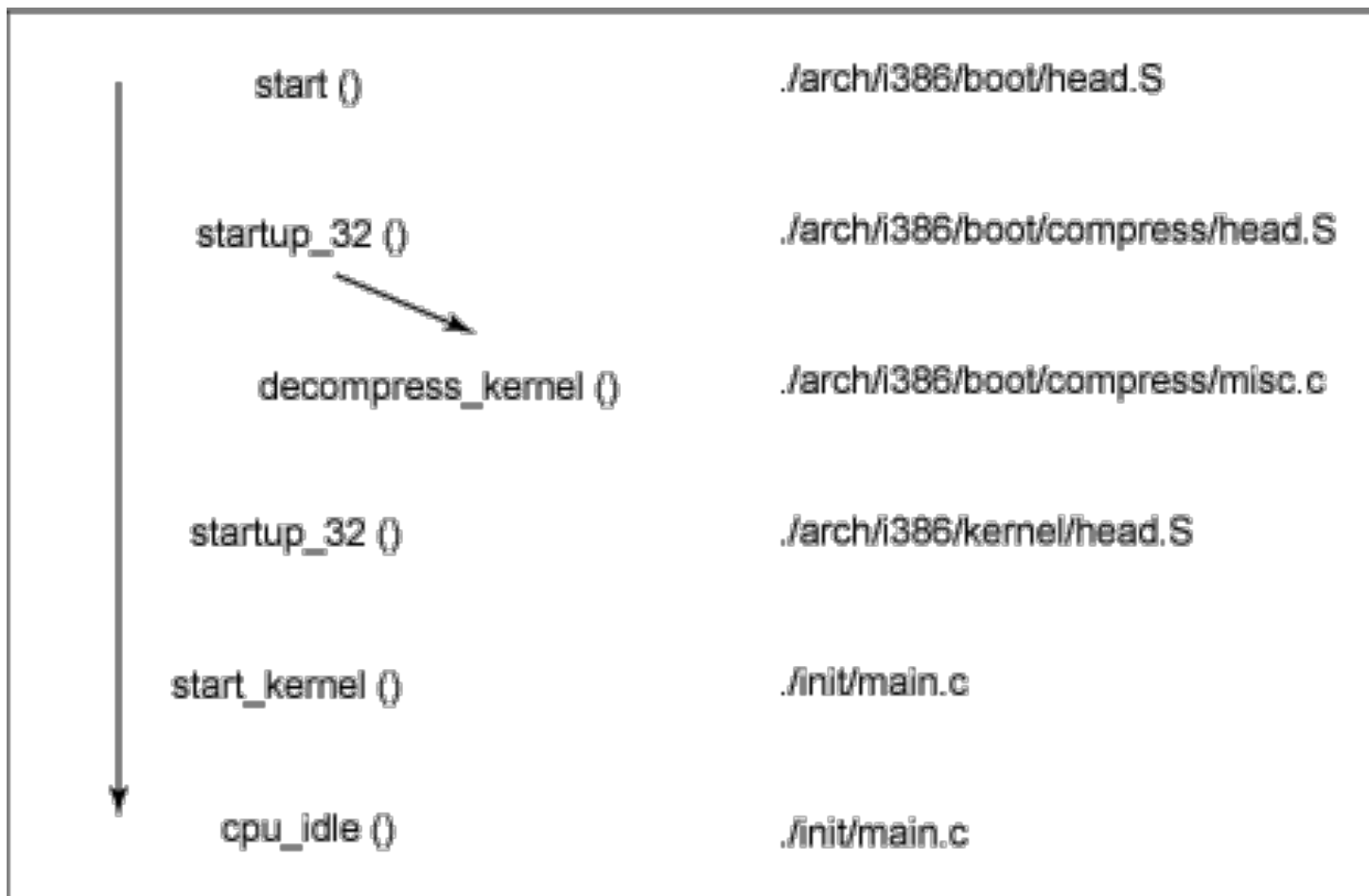
- 不断装载下一段可执行代码
 - 扇区拷贝
 - 支持文件系统
 - 设置内存
 - 解压缩
 - 切换CPU模式
 - ...

<https://en.wikipedia.org/wiki/Vmlinux>



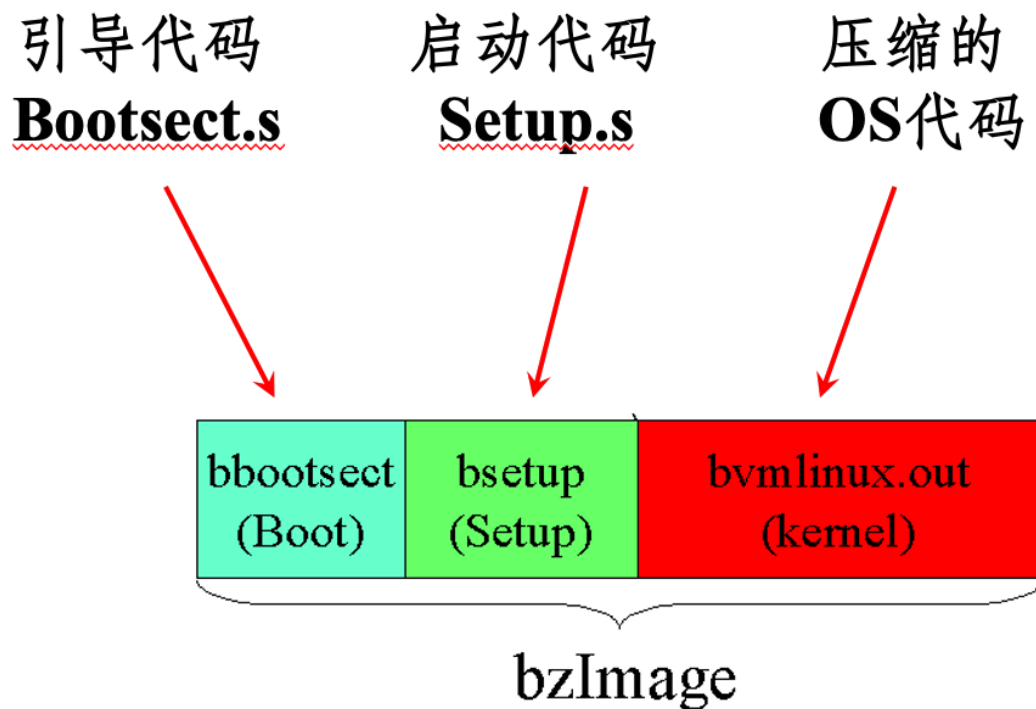
在Linux 2.6
内核中，
bootsect.S和
setup.S被整
合为
header.S

Major functions flow for Linux kernel boot

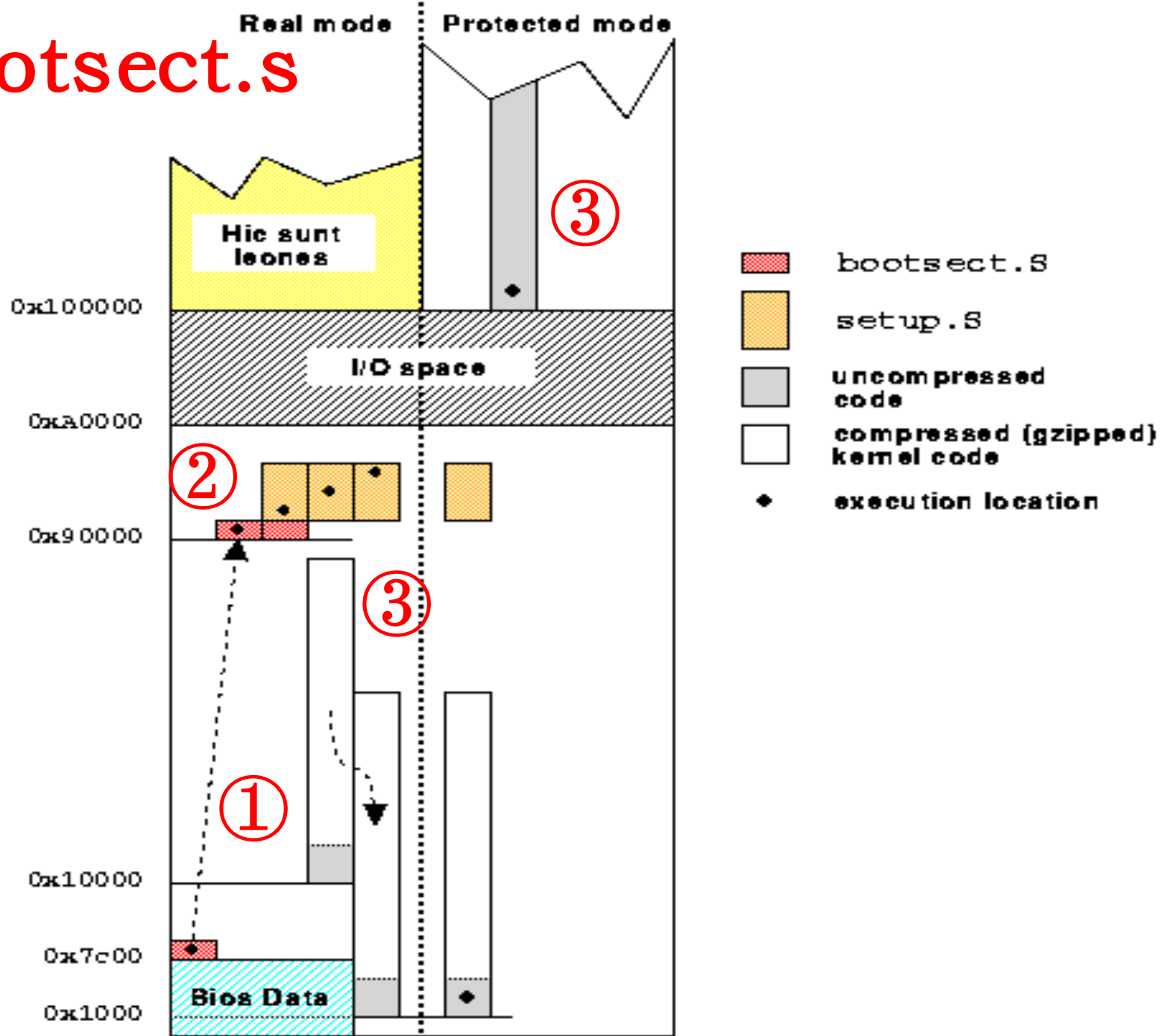


用于加载内核的关键文件

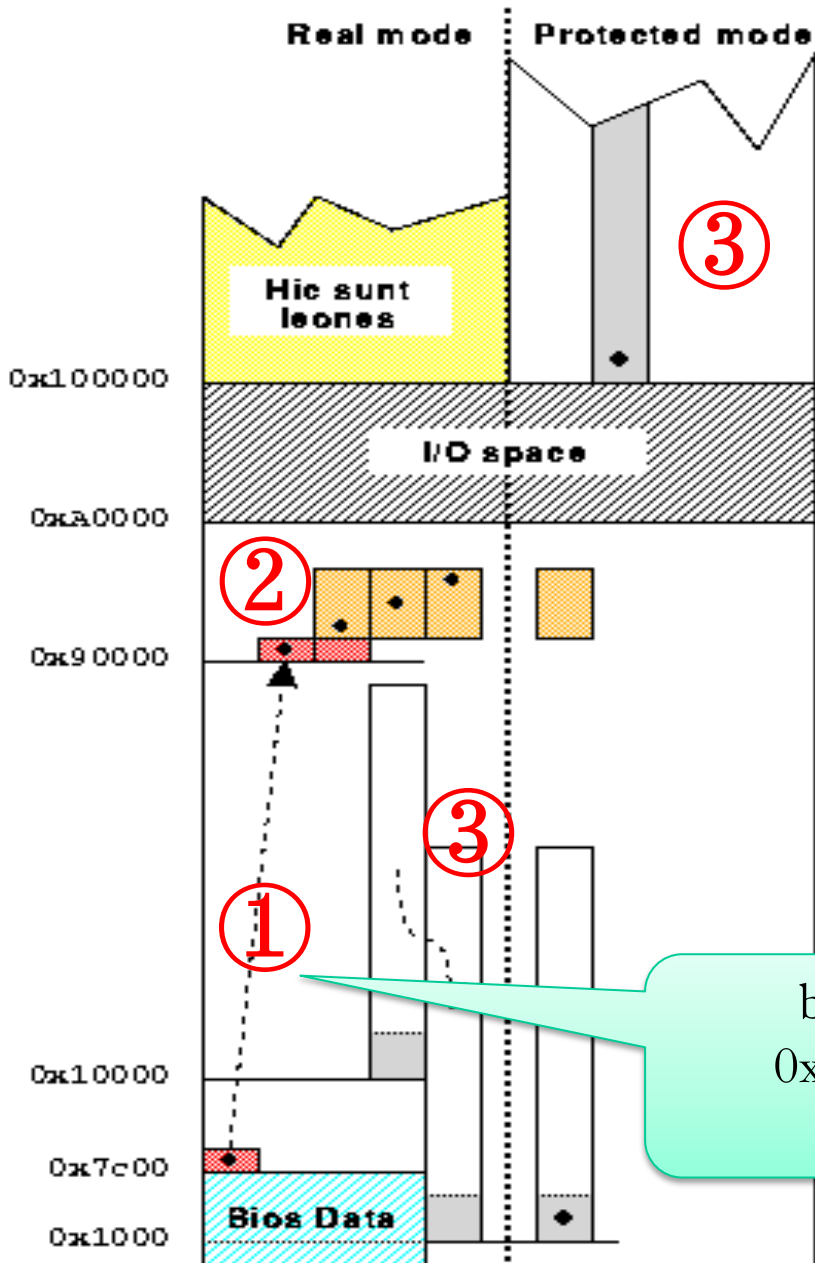
- bootsect.S :装载系统并设置系统启动过程的相关参数.
- setup.S :初始化系统及硬件,并切换至保护模式.
- video.S :初始化显示设备.



Bootsect.s

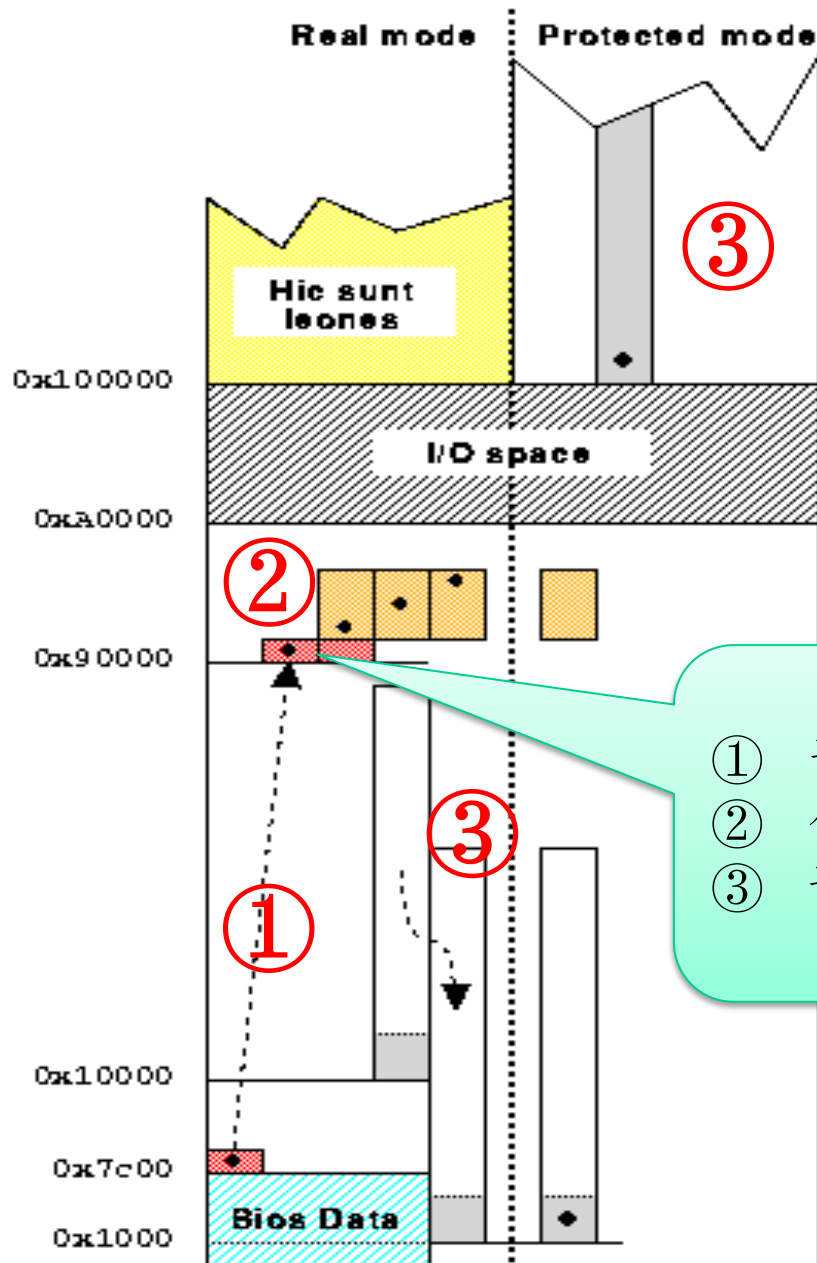


Bootsect.s



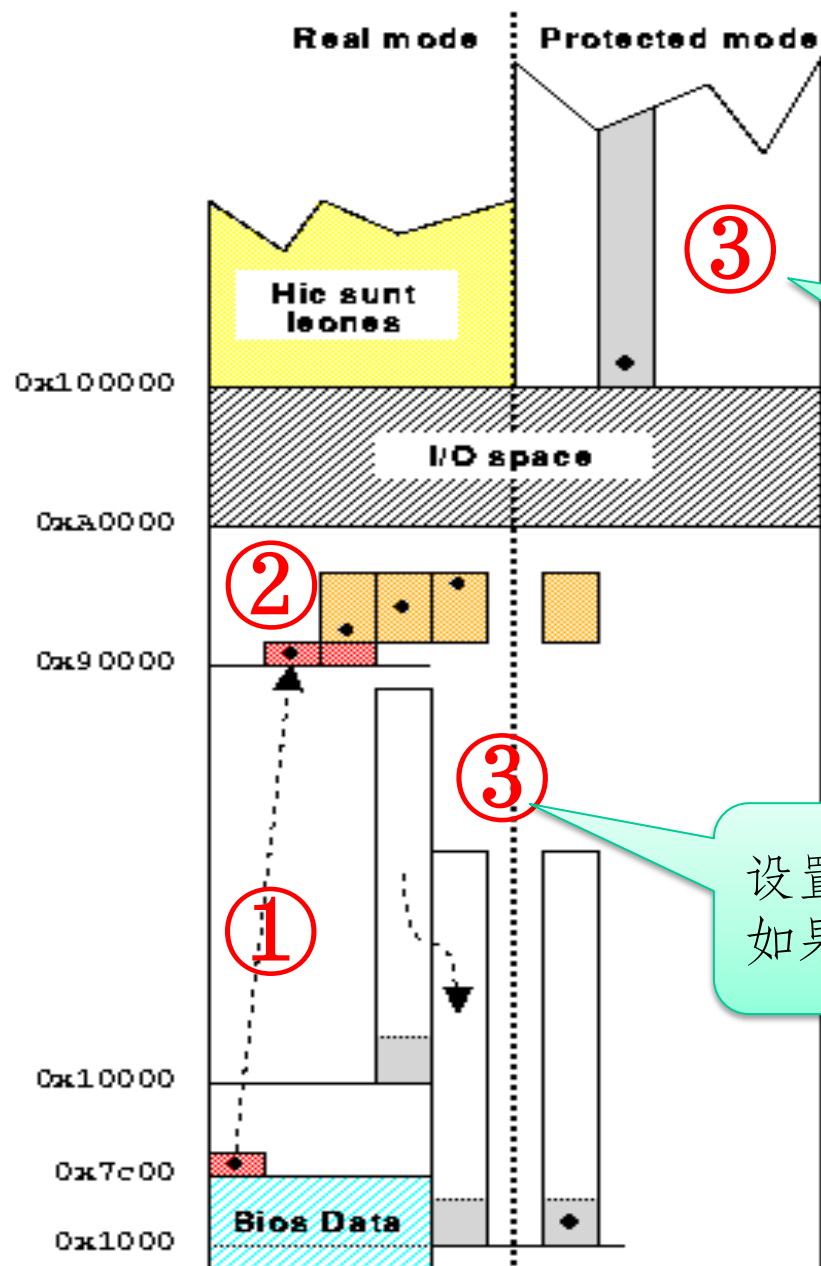
boot过程首先将自身从原始启动区0x7c00—0x7dff移至0x90000—0x901ff,并跳至下一条指令。

Bootsect.s



- ① 读引导扇区的后4个扇区到0x90200
- ② 代码主要调用引导阶段的函数，
- ③ 设置堆栈。

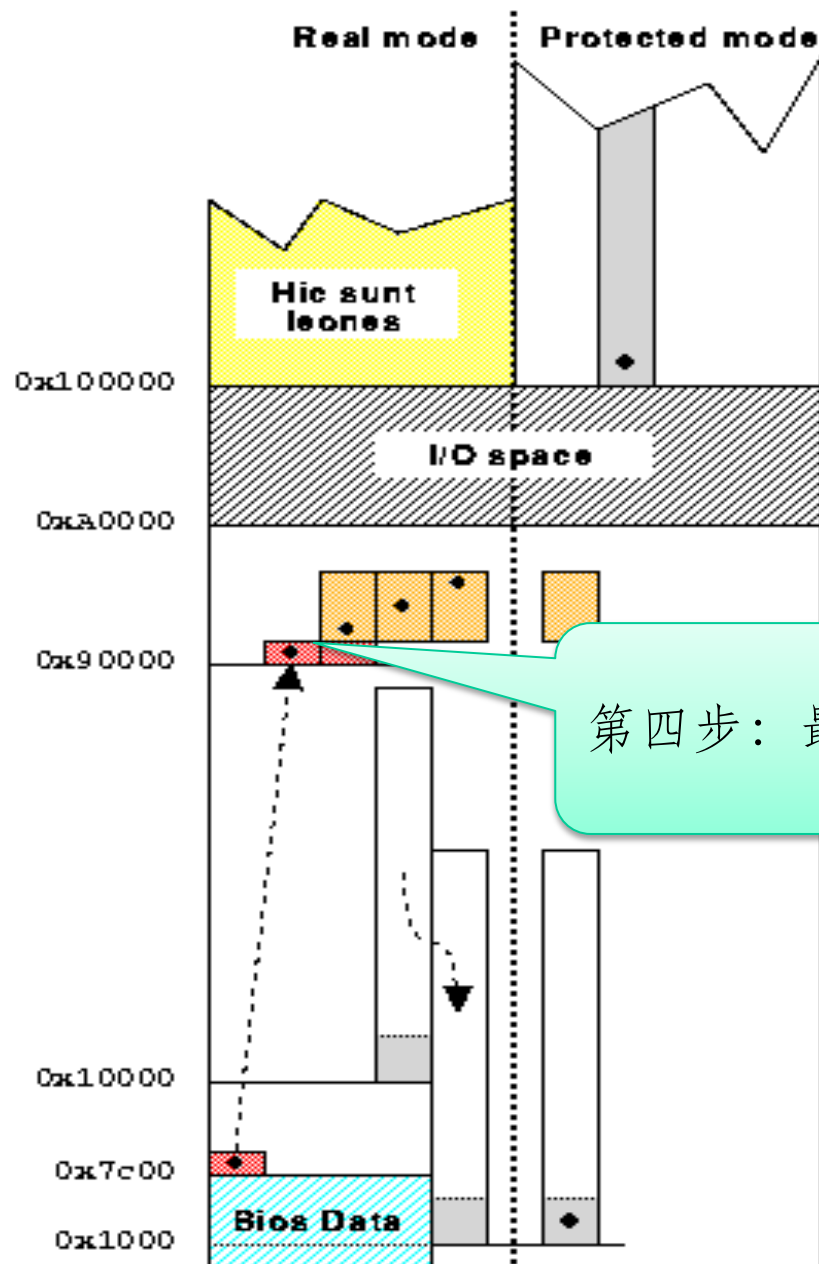
Bootsect.s



设置内核镜像：
如果是大内核，加载到0x100000
(1M以上高端内存)，

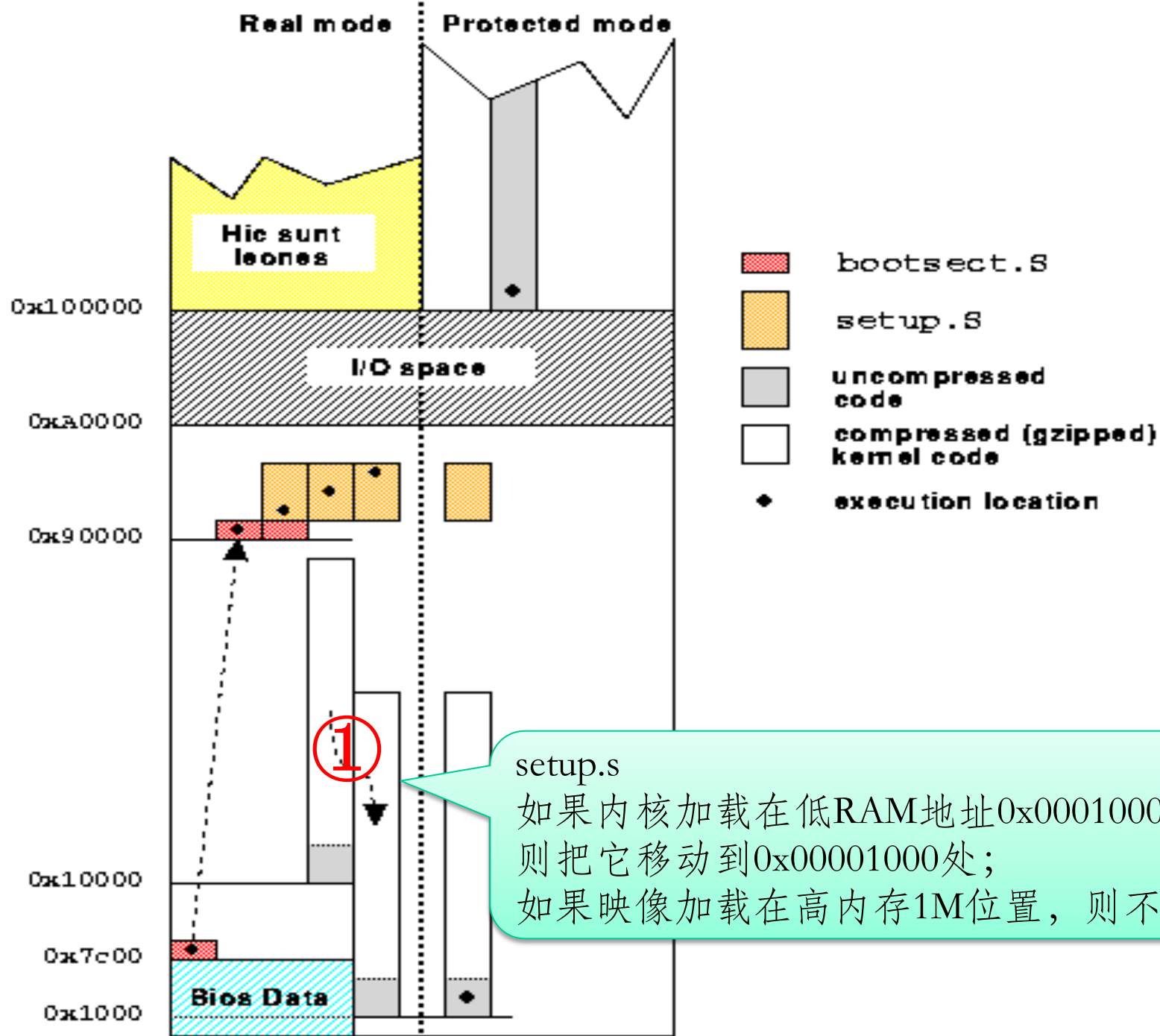
设置内核镜像：
如果是小内核，加载到0x10000 (64K低端内存)

Bootsect.s



Setup.S

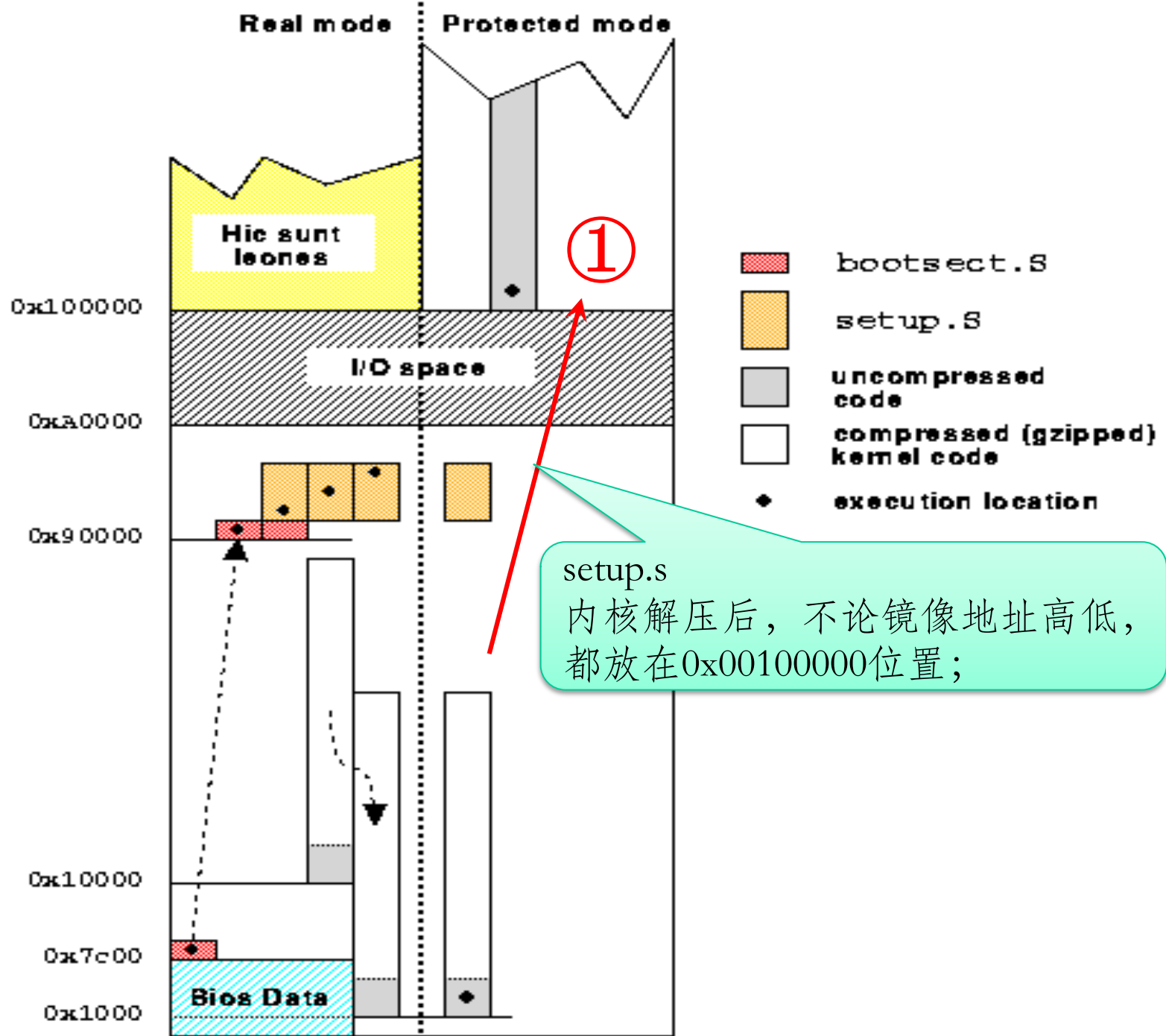
- **初始化硬件设备**。如：调用BIOS例程建立描述系统物理内存布局的表；设置键盘的重复延迟和速率；初始化显卡；检测IBM MCA总线、PS/2鼠标设备、APM BIOS支持等……。
- **为内核程序的执行建立环境**。如：启动位于8042键盘控制器的A20 pin。建立一个中断描述表IDT和全局描述表GDT表；重启FPU单元；对可编程中断控制器进行重新编程，屏蔽所有中断；**设置CR0状态寄存器的PE位使CPU从实模式切换到保护模式**，PG位清0，禁止分页功能等……；
- 跳转到startup_32()汇编函数，`jmp 0x100000, __BOOT_CS`，终于进入内核Head.S；



Head.S (第一个start_32()函数)

setup 结束后，该函数被放在 0x00001000 或者 0x00100000位置，该函数主要操作：

- 首先初始化段寄存器和临时堆栈；
- 清除eflags寄存器的所有位；
- 将_edata和_end区间的所有内核未初始化区填充0；
- ① 调用decompress_kernel()函数解压内核映像。首先显示“Uncompressing Linux...”信息，解压完成后显示“OK, booting the kernel.”。
- 跳转到0x00100000物理内存处执行；



Head.S (第二个start_32()函数)

- 解压后的映像开始于arch/i386/kernel/head.S 文件中的startup_32()函数，因为通过物理地址的跳转执行该函数的，所以相同的函数名并没有什么问题。该函数为Linux第一个进程建立执行环境，操作如下：
 - a) 初始化ds,es,fs,gs段寄存器的最终值；
 - b) 用0填充内核bss段；
 - c) 初始化swapper_pg_dir数组和pg0包含的临时内核页表
 - d) 建立进程0idle进程的内核模式的堆栈；
 -
 - x) 跳转到start_kernel函数，这个函数是第一个C编制的函数，内核又有了一个新的开始。

调用Start_kernel()

- 调用start_kernel()函数来启动一系列的初始化函数并初始化各种设备，完成Linux核心环境的建立：
 - 调度器初始化，调用sched_init();
 - 调用build_all_zonelists函数初始化内存区；
 - 调用page_alloc_init()和mem_init()初始化伙伴系统分配器；
 - 调用trap_init()和init_IRQ()对中断控制表IDT进行最后的初始化；
 - 调用softirq_init() 初始化TASKLET_SOFTIRQ和HI_SOFTIRQ；
 - Time_init()对系统日期和时间进行初始化；
 - 调用kmem_cache_init()初始化slab分配器；
 - 调用calibrate_delay()计算CPU时钟频率；

至此，Linux内核已经建立起来了，基于Linux的程序应该可以正常运行了。

启动第五步——用户层init依据inittab文件来设定运行等级

- 内核被加载后，第一个运行的程序便是 **/sbin/init**，该文件会读取 **/etc/inittab** 文件，并依据此文件来进行初始化工作。
- **/etc/inittab** 文件最主要的作用就是设定 **Linux** 的运行等级，其设定形式是 “**:id:5:initdefault:**”，这就表明 **Linux** 需要运行在等级 **5** 上。

Linux的运行等级

Linux的运行等级设定如下：

- 0：关机
- 1：单用户模式
- 2：无网络支持的多用户模式
- 3：有网络支持的多用户模式
- 4：保留，未使用
- 5：有网络支持有**X-Window**支持的多用户模式
- 6：重新引导系统，即重启

启动第六步——init进程执行 `rc.sysinit`

- 在设定了运行等级后，Linux系统执行的第一个用户层文件就是`/etc/rc.d/rc.sysinit`脚本程序，它做的工作非常多，包括设定PATH、设定网络配置（`/etc/sysconfig/network`）、启动swap分区、设定`/proc`等等。
- 如果你有兴趣，可以到`/etc/rc.d`中查看一下`rc.sysinit`文件，非常复杂。

启动第七步——启动内核模块

- 具体是依据 **/etc/modules.conf** 文件或 **/etc/modules.d** 目录下的文件来装载内核模块。

启动第八步——执行不同运行级别的脚本程序

- 根据运行级别的不同，系统会运行 **rc0.d** 到 **rc6.d** 中的相应的脚本程序，来完成相应的初始化和启动相应的服务。

启动第九步——执行/etc/rc.d/rc.local

- 你如果打开了此文件，里面有一句话，读过之后，你就会对此命令的作用一目了然：

This script will be executed *after* all the other init scripts.

You can put your own initialization stuff in here if you don't

want to do the full Sys V style init stuff.

- **rc.local**就是在一切初始化工作后，**Linux**留给用户进行个性化的地方。你可以把你想要设置和启动的东西放到这里。

启动第十步——执行/bin/login程序，进入登录状态

- 此时，系统已经进入到了等待用户输入 **username** 和 **password** 的时候了，你已经可以用自己的帐号登入系统了。

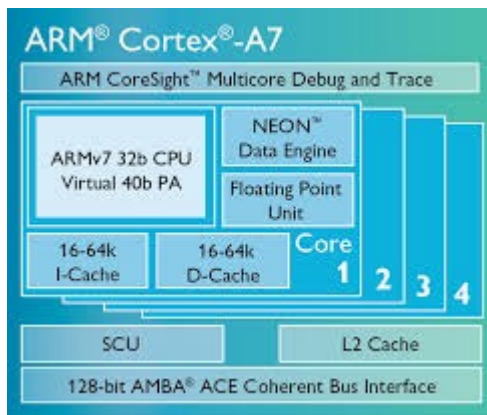
漫长的启动过程结束了.

其实在这背后，还有着更加复杂的底层函数调用，等待着你去研究……

课堂上的内容就算抛砖引玉了

结合实验

- 阅读《See MIPS Run》，了解MIPS启动过程
- 查阅资料，了解ARM（树莓派）启动过程



思考：为什么操作系统启动慢

- 写一个不少于1页A4纸的调研报告
- 给出对OS启动过程的分析，指出最耗时的启动过程是什么？
- 现有的优化措施有哪些？
- 思考并给出你的优化建议。