Demystifying the Kernel Bootstrap Process

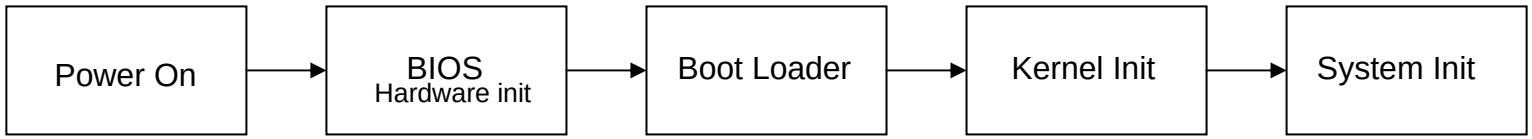
Claudia Salzberg

IBM Linux Technology Center

Overview

- Highlight the multiple levels of the bootstrap process.
- Explain the purpose of each of these levels.
- Illustrate how these levels interleave and interrelate

Booting Process Overview



- BIOS

  Hardware initialization upon PC power-up. This involves a basic memory check, the initialization of key devices, and the determination of bootable devices.

- Boot Loader

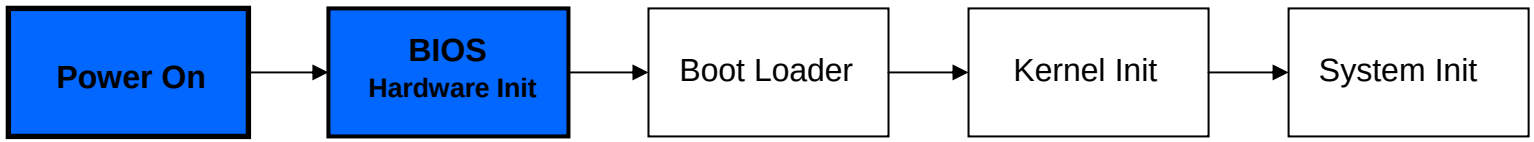
  Identifies bootable images, places the selected image into memory and jumps to its point of entry for execution.

- Kernel Initialization

  Linux kernel takes control of system. It initializes its subsystems, probes and initializes for particular devices. Jumps to init process

- System Initialization

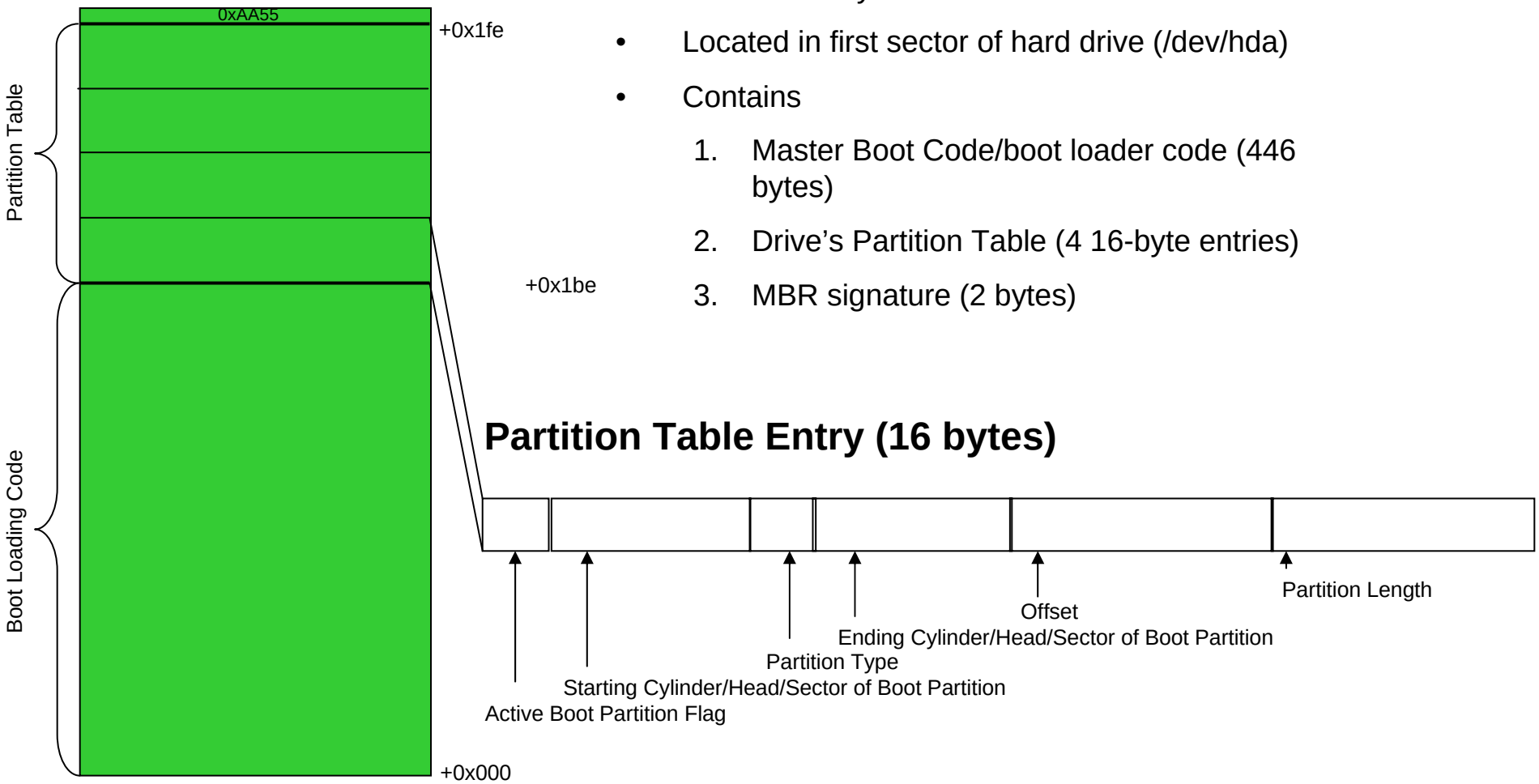
Hardware Power On



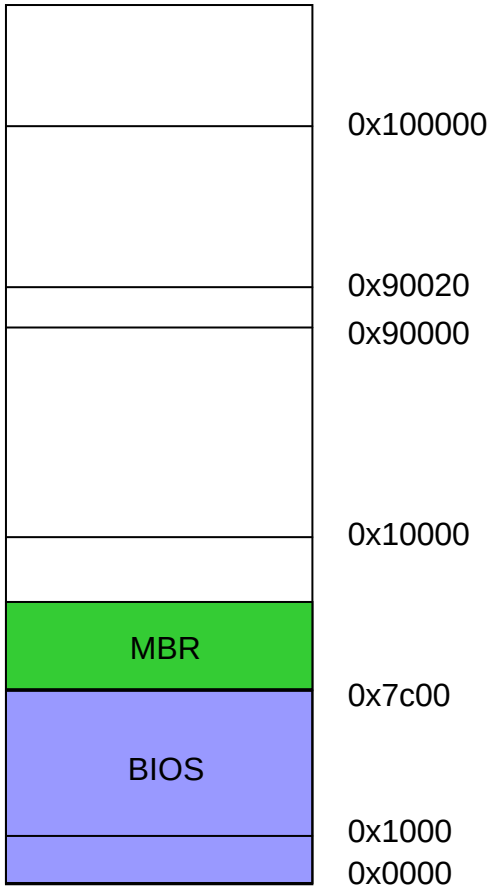
- Main memory is empty.
- CPU begins execution by accessing a pre-defined address in ROM intended for the initial boot process (0x000F:FFF0).
- BIOS ROM (firmware) is loaded to main memory
- BIOS performs basic tests on the hardware
  - Initializes the keyboard, video card, system board, memory, and identifies additional I/O devices.
  - Examines first sector of bootable disk for the MBR
  - Copies first 512 bytes from boot device (boot sector) to 0x7c00 (tests for valid signature)
- MBR determines active boot partition and jumps to it
- Boot Loader is located in active boot partition (can also be elsewhere)

A Closer Look at the MBR

- MBR is 512-byte boot sector in PC architecture
- Located in first sector of hard drive (/dev/hda)
- Contains
  1. Master Boot Code/boot loader code (446 bytes)
  2. Drive's Partition Table (4 16-byte entries)
  3. MBR signature (2 bytes)



View of Memory over Time



Current Addressing Mode:

Real Mode

Protected Mode

Jump to Boot Loader: LILO

- What is a Boot Loader?
  - A limited functionality program dedicated to spanning the gap between firmware routines and the full fledged operating system.
- LILO is one of the earliest boot-loading programs
- Located in MBR (or root partition)
- Makes use of BIOS functions (Interrupt calls) to load sectors
- Basic Steps:
  - Load operating system into memory
  - Load ramdisk (optional)
  - Pass kernel arguments to kernel
  - Transfer execution to kernel

Jump to Boot Loader: LILO

- Load Kernel into Memory
  - Copies kernel image referenced in /etc/lilo.conf from hard drive using BIOS services
  - Loads kernel image to 0x90000 (using firmware functions):
    - 0x900000 (bootsect sector)
    - 0x900200 (setup sector)
    - 0x1000000 (compressed image)
  - Jumps to label start_of_setup @ 0x90200
- Alternative to bootloader – using bootsect.S

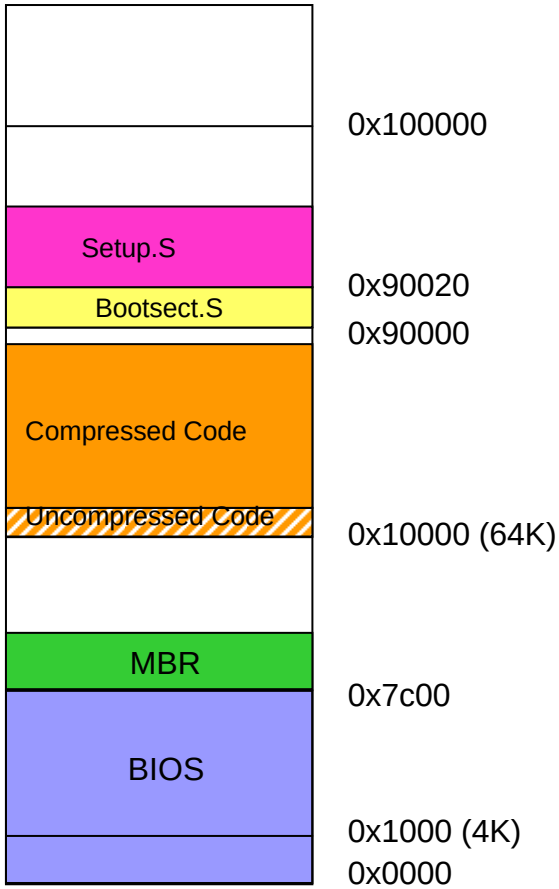
Kernel Image Topology

- Building the kernel Image:
  - User input of **make zImage** or **make bzImage**
  - All C and assembly source files are compiled and linked into **vmlinux**
  - nm vmlinux => System.map
  - **arch/i386/boot**

Type of Kernel Image	IMAGE_OFFSET	Preprocessor Flags
bzImage	0x100000	-D__BIG_KERNEL__
zImage	0x1000	

  - bootsect.S is assembled into bootsect.s and converted into raw binary form
  - setup.S is assembled into setup.s and converted into raw binary form
  - **arch/i386/boot/compressed**
    - Remove .note and .comment ELF sections from vmlinux
    - Gzip vmlinux
    - Compile compression routines in head.S and misc.c and link into vmlinux.out
  - **arch/boot/tools/build** [-b] bootsect setup system [rootdev] [> image]
    - Concatenates bootsect, setup, and compressed/vmlinux.out into **zImage**

View of Memory over Time

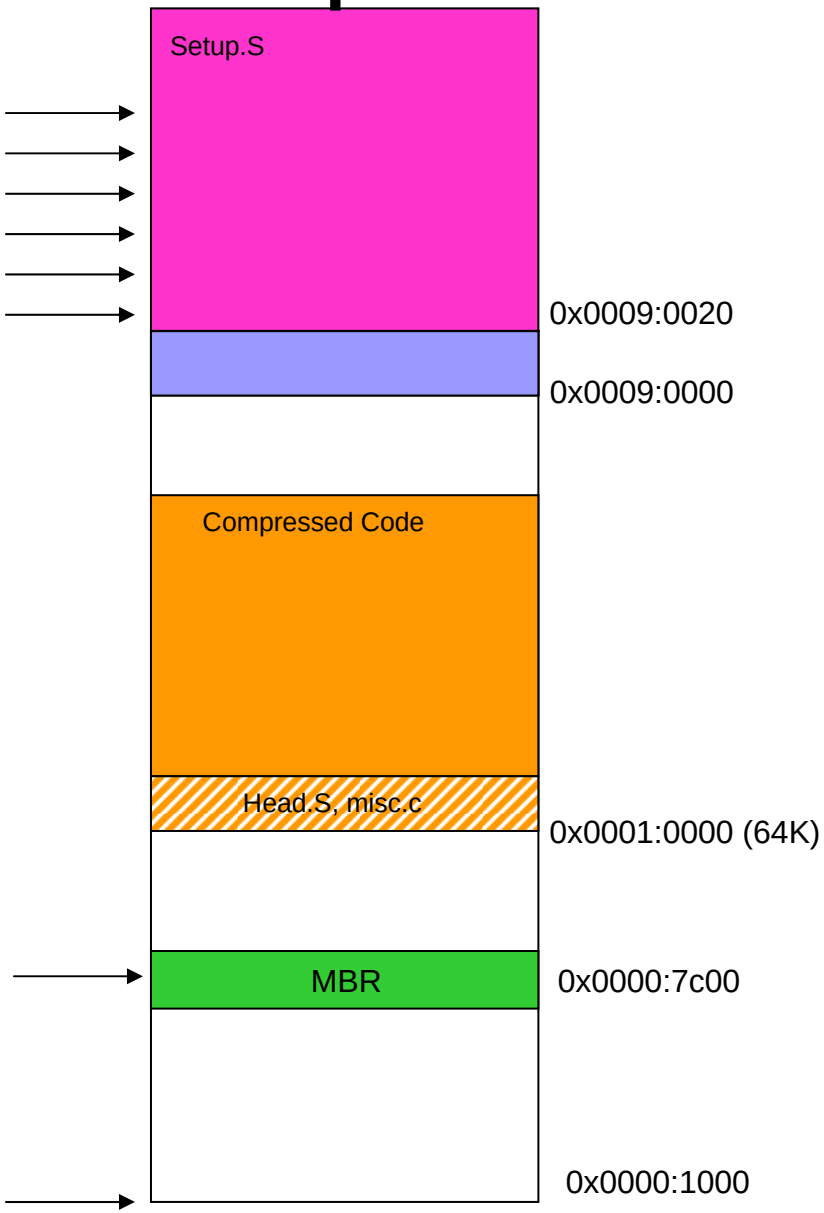


Current Addressing Mode:

Real Mode

Protected Mode

Setup.S Execution



Jump to 0x0009:0020

Get system data from BIOS and generate memory map

Initialize hardware

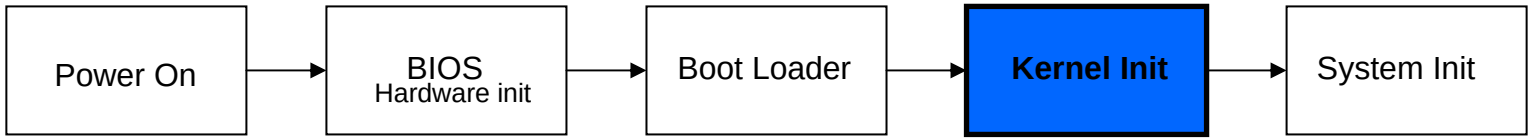
Move compressed kernel

Set up provisional IDT and GDT

Change from real to protected mode

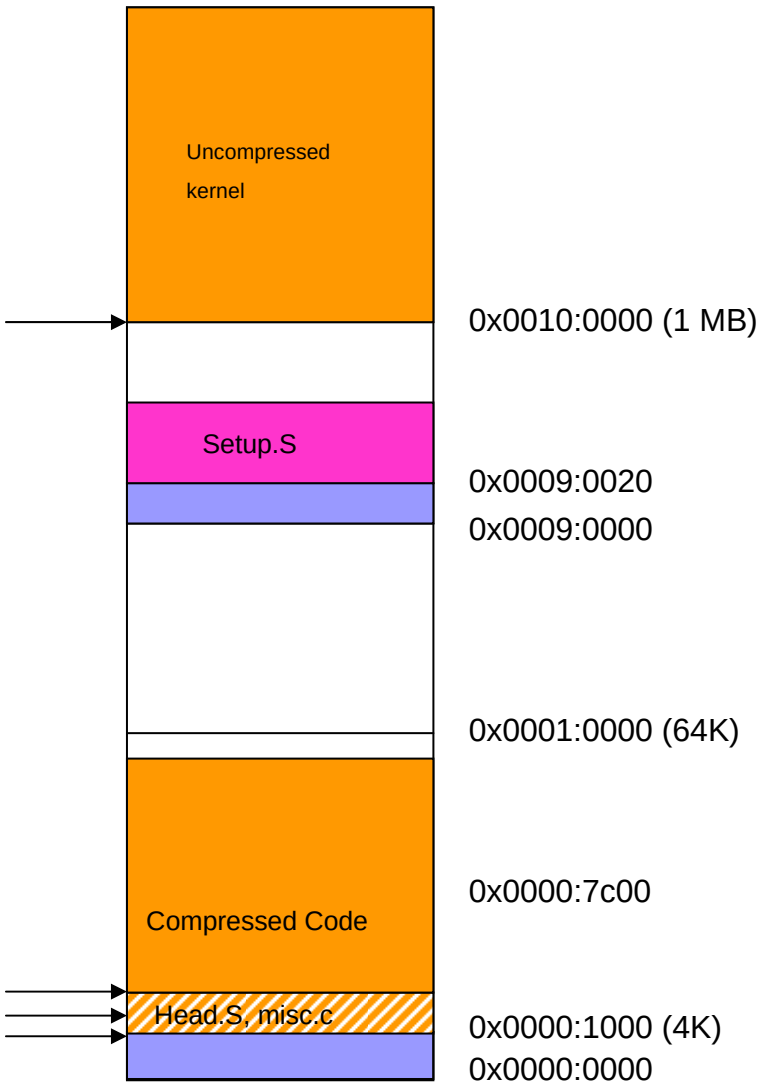
Jump to head.S startup_32()

Kernel Initialization



- The jump to startup_32 (head.S)
  - Initialize page tables
- The jump to start_kernel() (init/main.c)
  - Initialize all the kernel subsystems
  - Execute Init()

Head.S execution - startup_32()



Initialize page tables, enable paging

Set-up stack (zero out BSS)

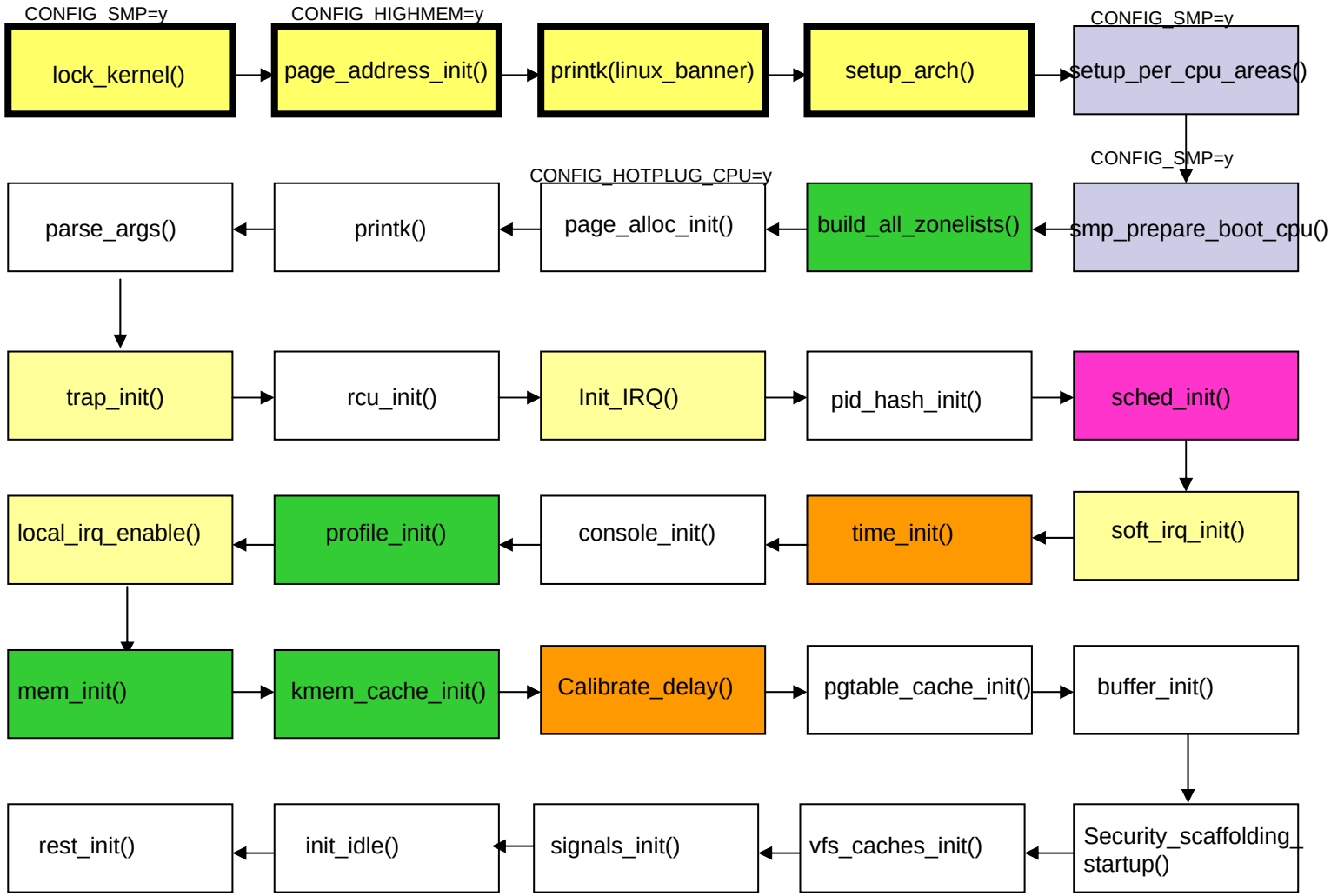
Call decompress_kernel()

Uncompressing Linux..

Ok, booting the kernel.

Jump to start_kernel()

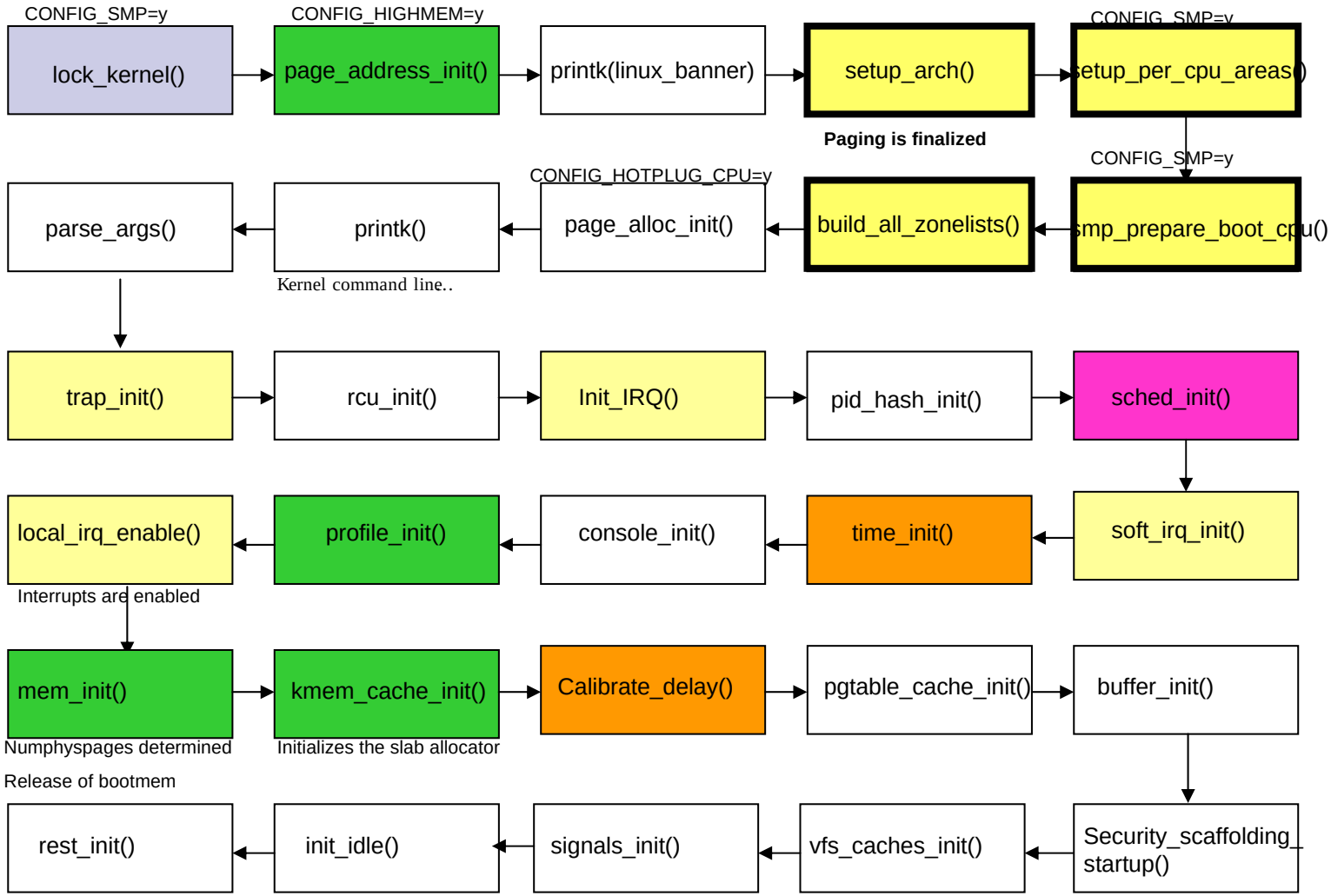
High level initialization: start_kernel() to init()



setup_arch(&command_line)

- Takes in a pointer to Linux command-line data entered at boot time
- Initializes architecture-specific subsystems
- Grabs data from boot time structures
- Identifies the processor
- Grabs and performs early parsing of boot time parameters
- Calls paging_init()

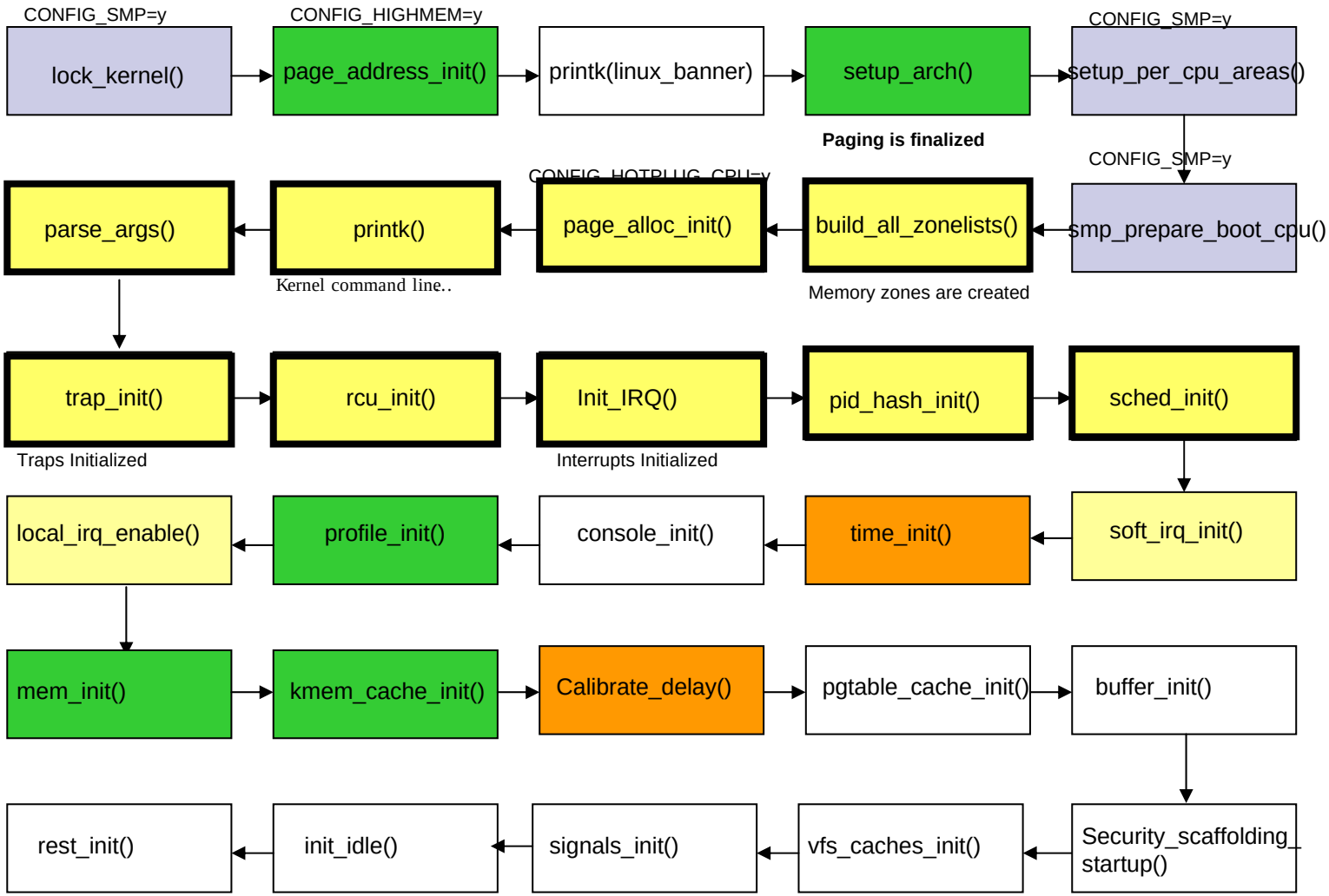
High level initialization: start_kernel() to init()



build_all_zonelists()

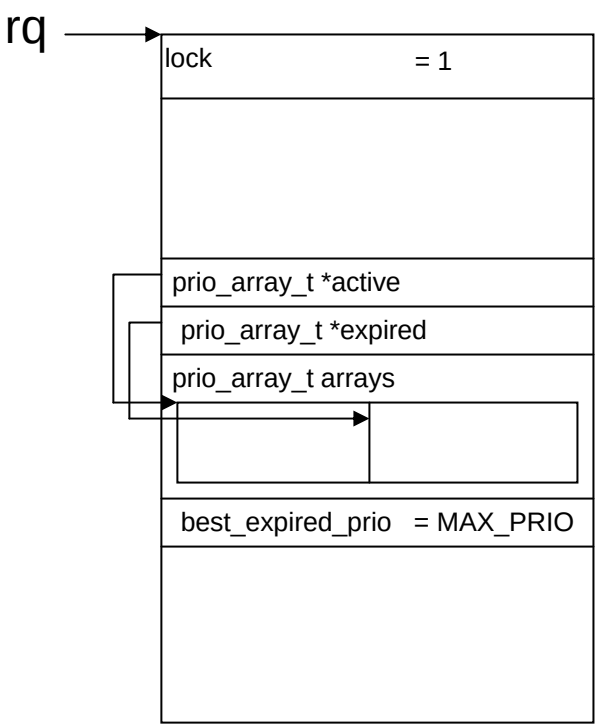
- Splits up memory into the three zones:
  - ZONE_DMA
  - ZONE_NORMAL
  - ZONE_HIGHMEM
- zones: linear separations of physical memory that are used to address hardware limitations

High level initialization: start_kernel() to init()



sched_init()

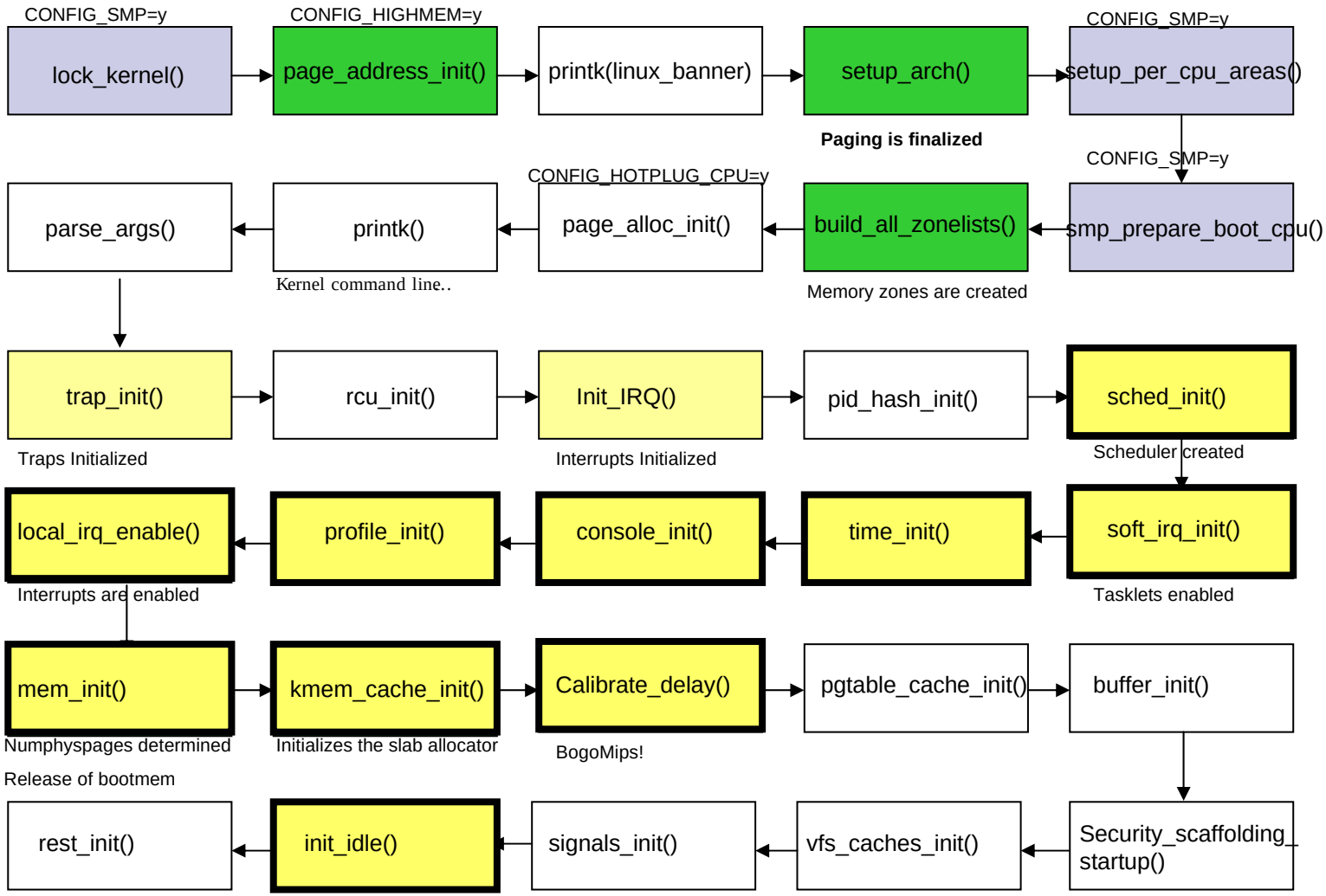
- Routine to initialize scheduler
- Each CPU's runqueue is initialized (active queue, expired queue, and spinlock)



softirq_init()

- Prepares the CPU to accept notification from tasklets
- Kernel response time upon interrupts are kept small by splitting the interrupt-generated task into tasklets that can be queued and handled at a different time.
- Event:
  - Interrupt is generated by some device
  - The interrupt handler schedules a tasklet (softirq) and exits
  - Eventually, tasklet is executed.

High level initialization: start_kernel() to init()

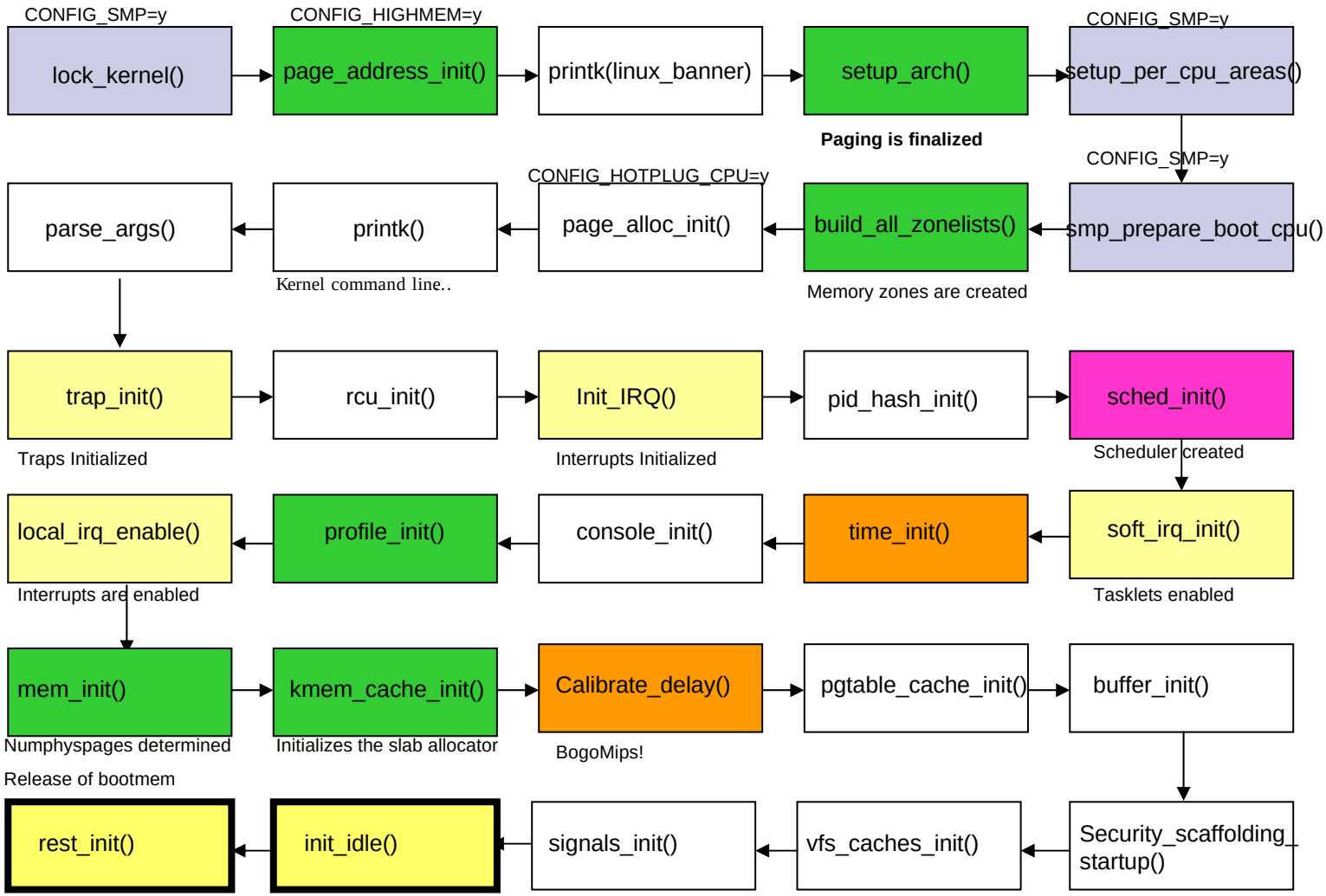


rest_init()

```
static void noinline rest_init(void)
{
        kernel_thread(init, NULL, CLONE_FS | CLONE_SIGHAND);
        unlock_kernel();
        cpu_idle();
}
```

- Creates the init thread
- Removes the kernel lock
- Calls the idle thread

High level initialization: start_kernel() to init()

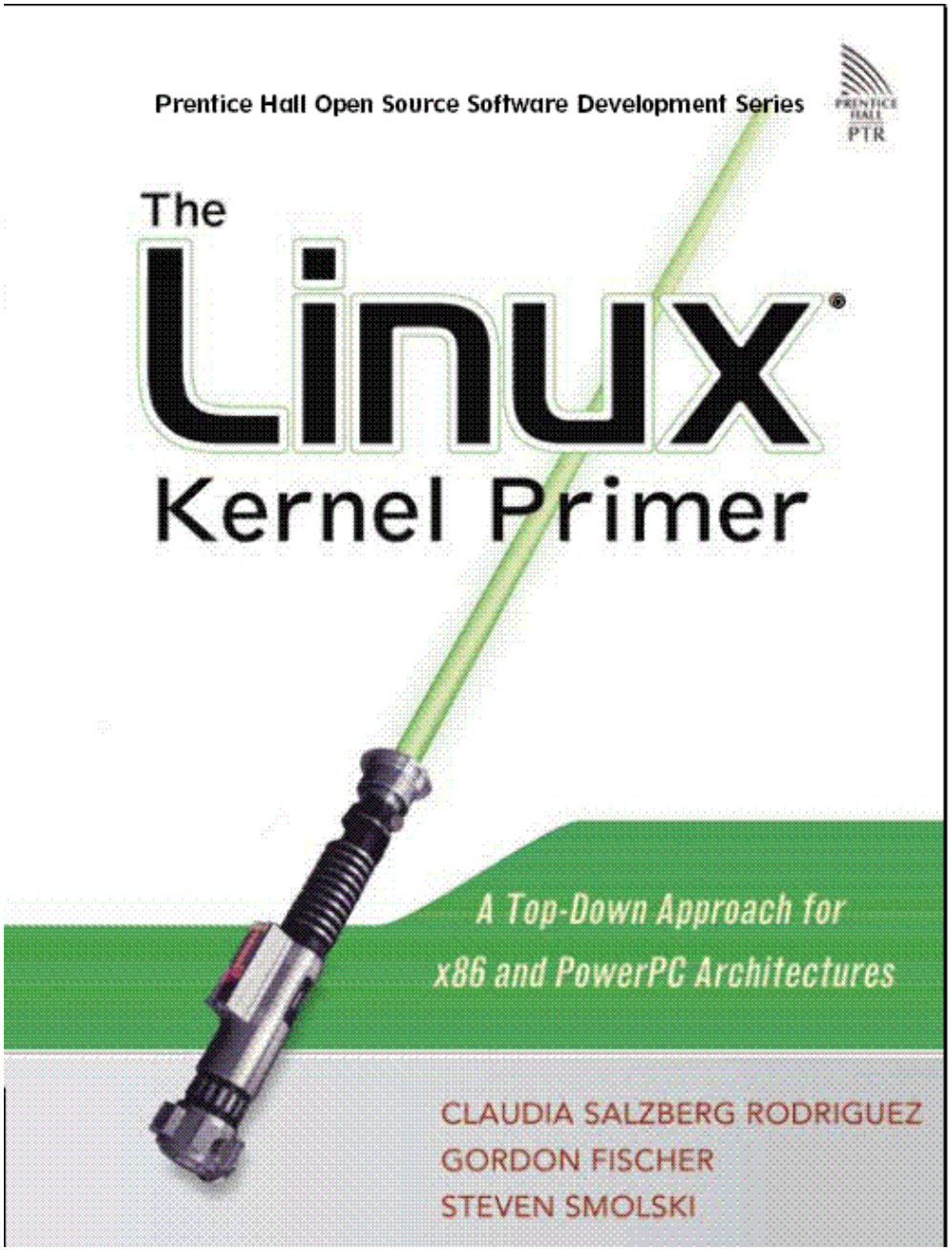


The End of Kernel Initialization: init()

- Set to reap any thread whose parent has died
- Initializes
  - Driver model and subsystems involved in driver support
  - Sysctl interface
  - Network socket interface
  - Work queue support
  - Prepares the namespace for the filesystem hierarchy
  - Mounts /dev, nfs root mount, root device mounting
  - Calls free_initmem() to clear memory segments with __init.
  - `run_init_process("/sbin/init");`

Lessons Learned

- The boot process has multiple steps each of which is dependent on its predecessor
- Many structures and functions are created that are only temporary


Prentice Hall Open Source Software Development Series
PRENTICE HALL PTR
The
Linux®
Kernel Primer
A Top-Down Approach for
x86 and PowerPC Architectures
CLAUDIA SALZBERG RODRIGUEZ
GORDON FISCHER
STEVEN SMOLSKI