Booting x86_64

Advanced Operating Systems

Overview

- PC boot sequence
- OpenLSD booting walkthrough

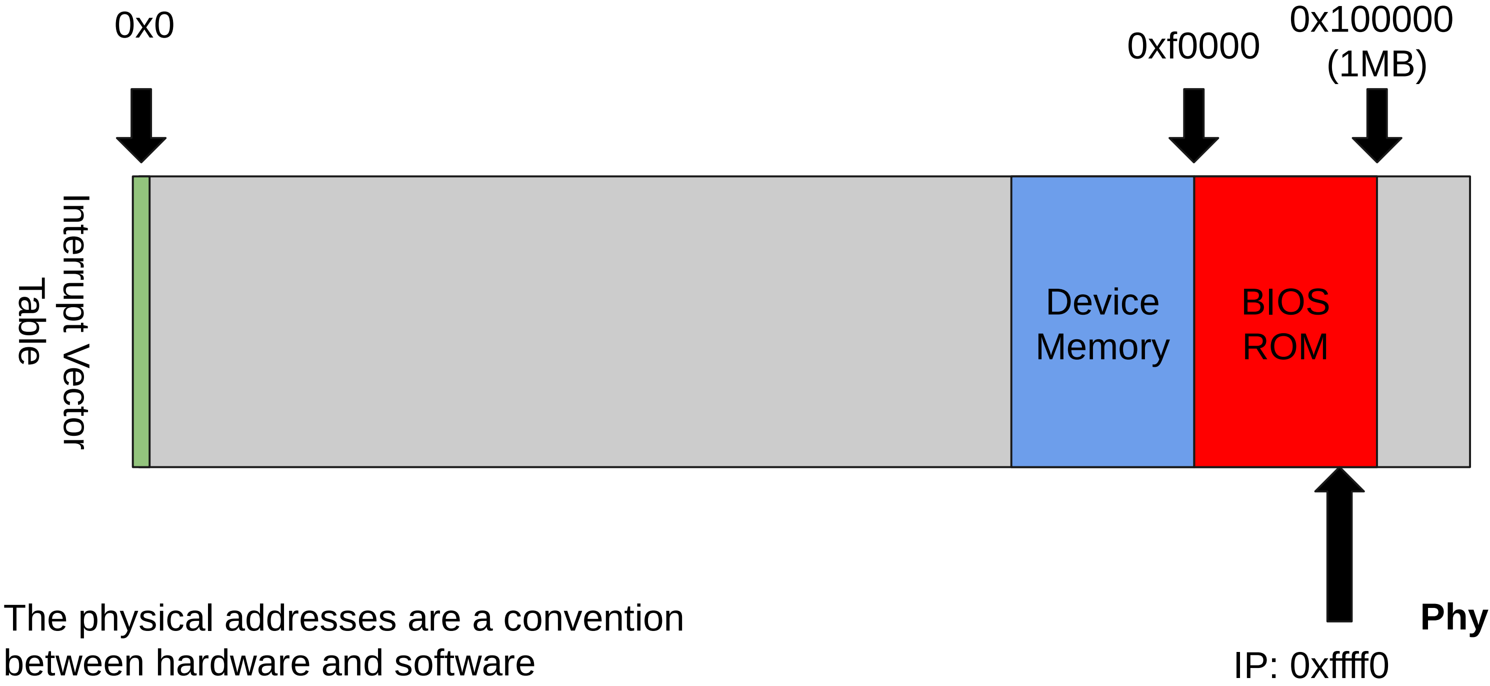
Booting x86_64

- Complicated and hairy
- Lots of legacy “things” to take care of
- Transitioning between CPU “modes”
  - Real mode (16 bit)
  - Protected mode (32 bit)
  - Long mode (64 bit)
- Memory addressing modes for each mode

Power on

- CPU executes code from ROM
- Load platform firmware
  - e.g. BIOS, UEFI, Coreboot, OpenFirmware
- Initializes memory and other devices
- Loads boot code into memory
- Executes boot code

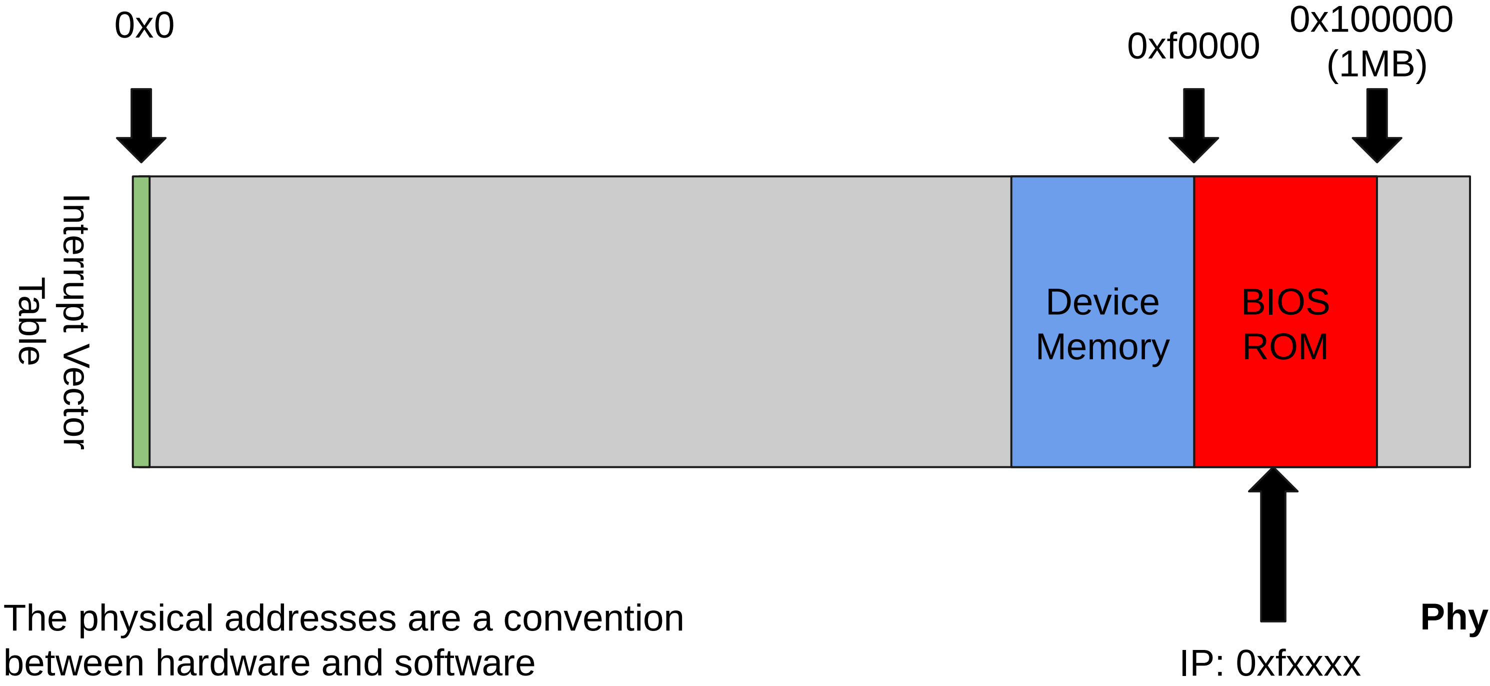
Memory layout



Physical Memory

The physical addresses are a convention between hardware and software

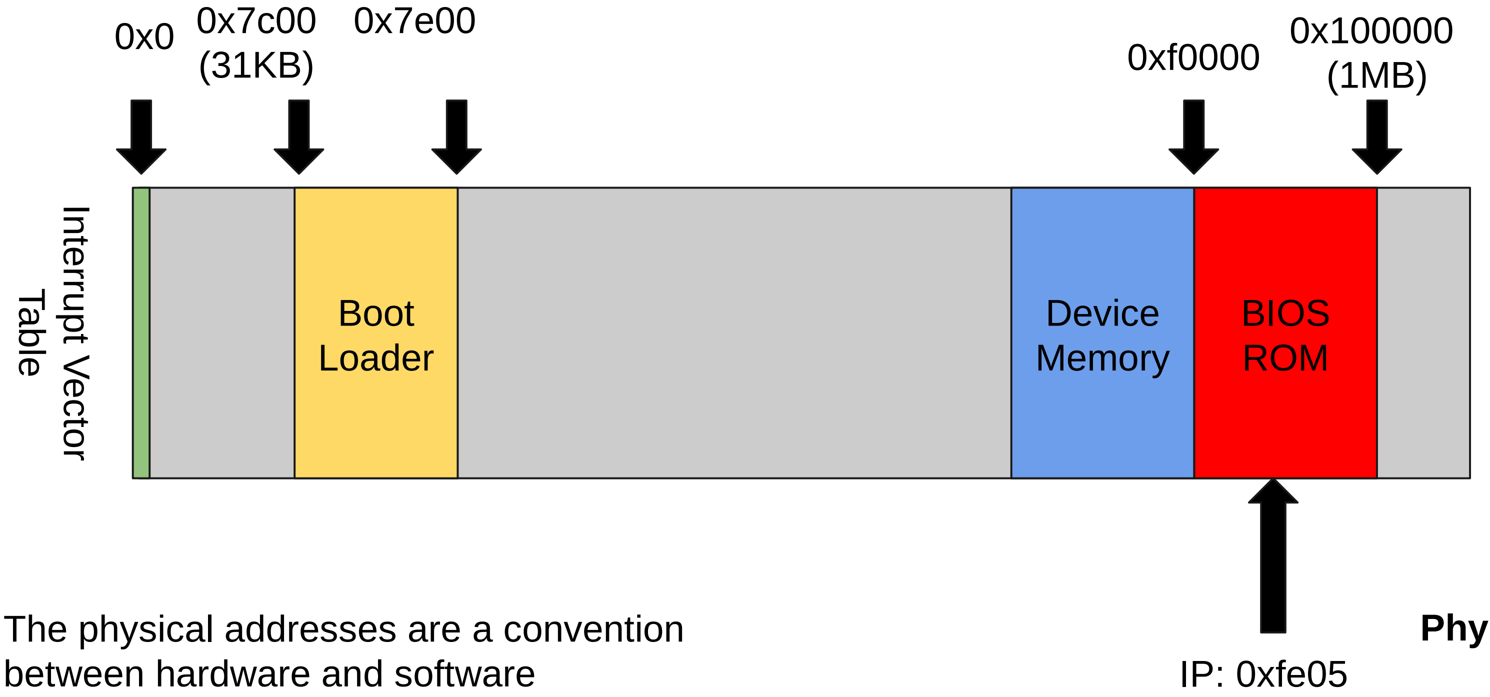
Memory layout



The physical addresses are a convention between hardware and software

Physical Memory

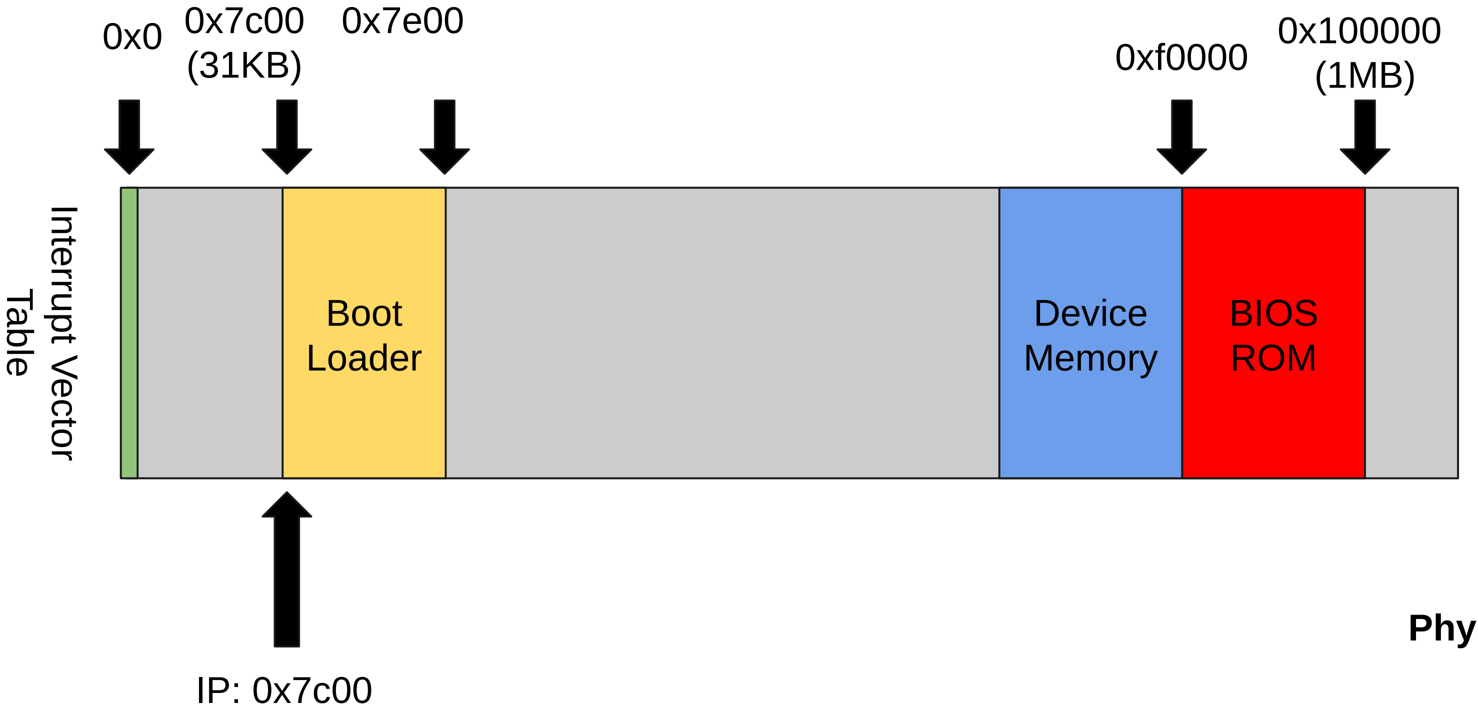
Memory layout



The physical addresses are a convention between hardware and software

Physical Memory

Memory layout

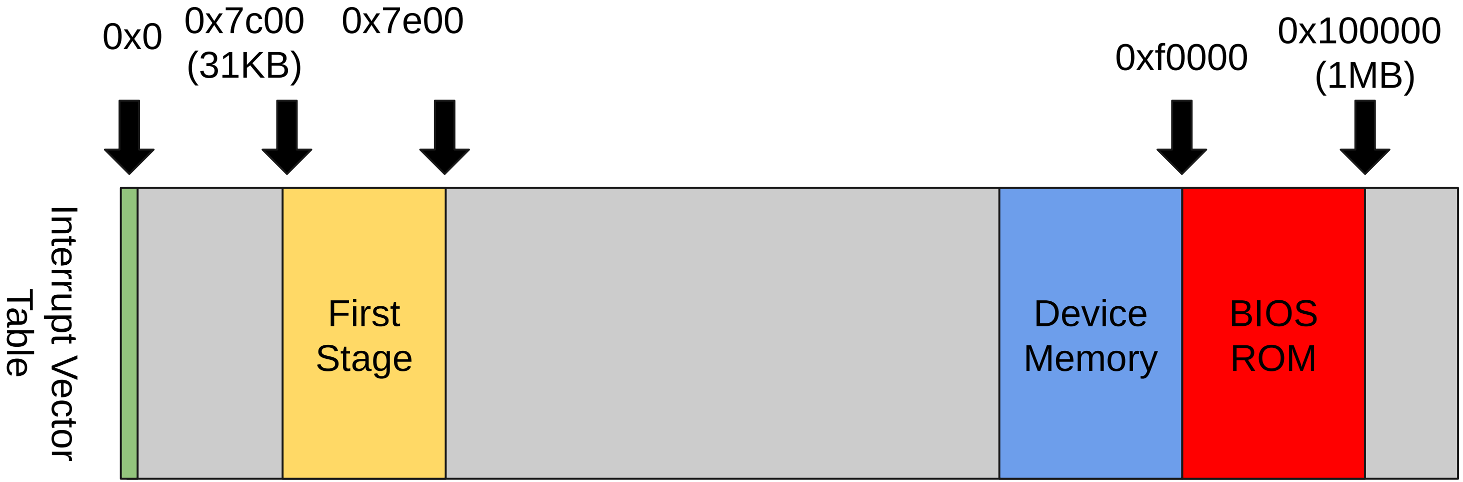


Physical Memory

Two-stage bootloader

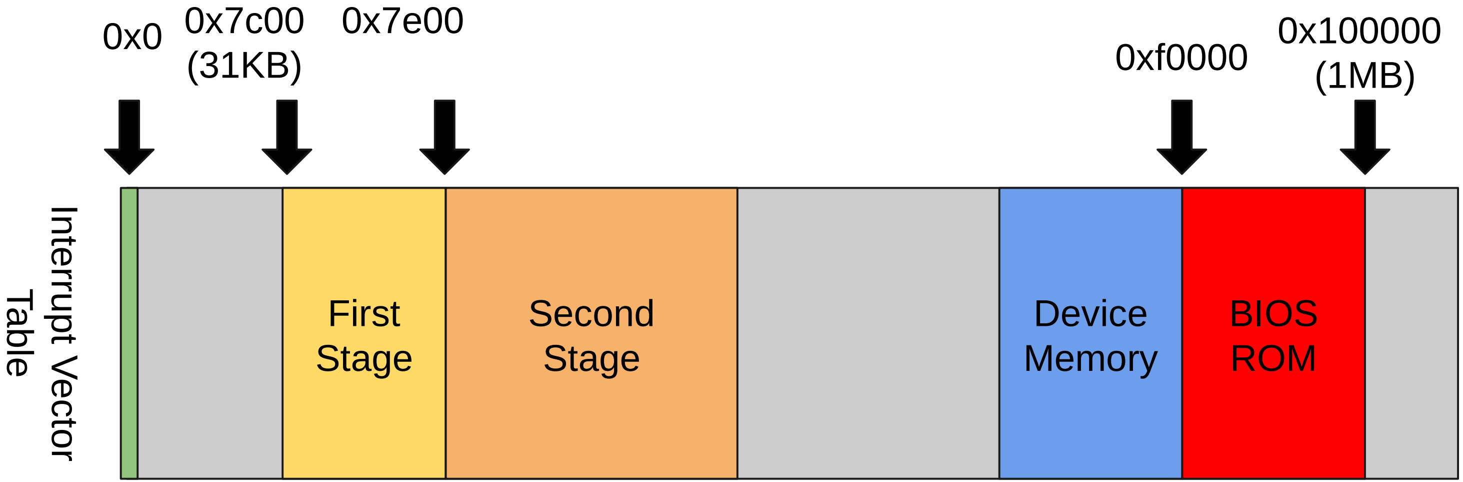
- BIOS only loads first disk sector
- A disk sector is at least 512 bytes
- Split up boot loader into two stages
- First stage loads the second stage

OpenLSD: boot/boot1.S



Physical Memory

OpenLSD: boot/boot1.S



0x0

0x7c00
(31KB)

0x7e00

0xf0000

0x100000
(1MB)

Interrupt Vector Table

First Stage

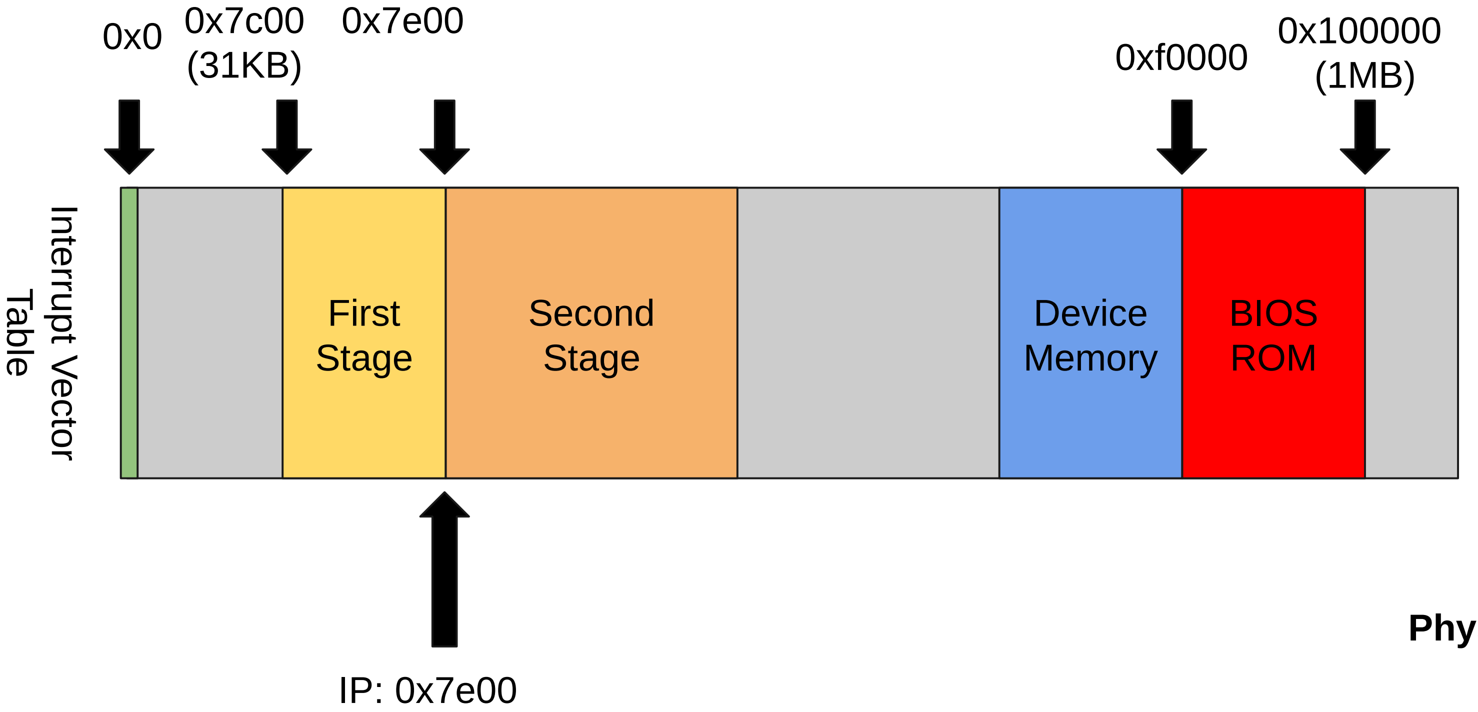
Second Stage

Device Memory

BIOS ROM

Physical Memory

OpenLSD: boot/boot2.S



Physical Memory

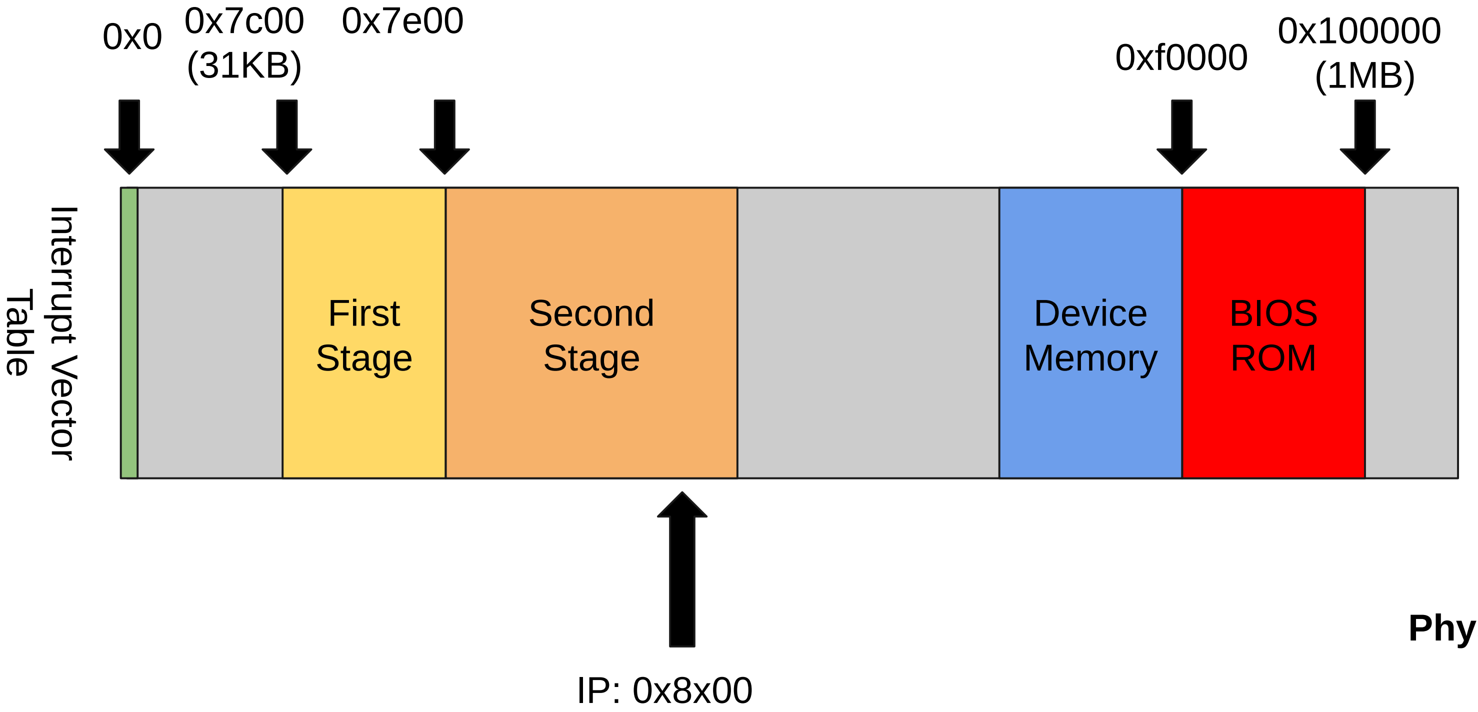
Memory map

- Not all memory is available to us yet:
  - `int 0x15`; `eax = 0xe820` (interrupt to BIOS)
  - Each entry describes a region of physical memory
- Bootloader does this for you :)
  - `kmain(struct boot_info *)`

Loading the kernel

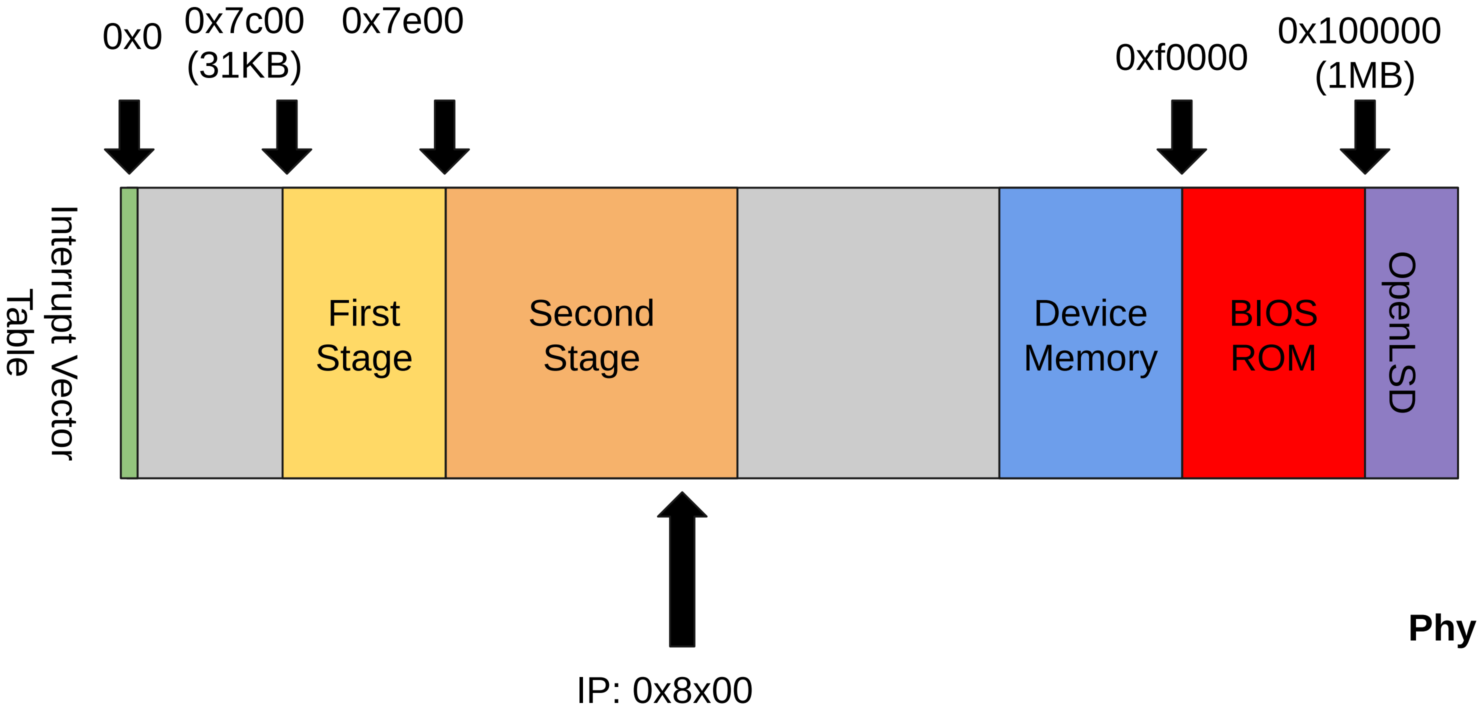
- OpenLSD uses the ELF binary format
- The kernel follows the boot loader
- After setting up protected mode, the boot loader reads the kernel into memory
- And jumps to the kernel entry function

OpenLSD: boot/main.c



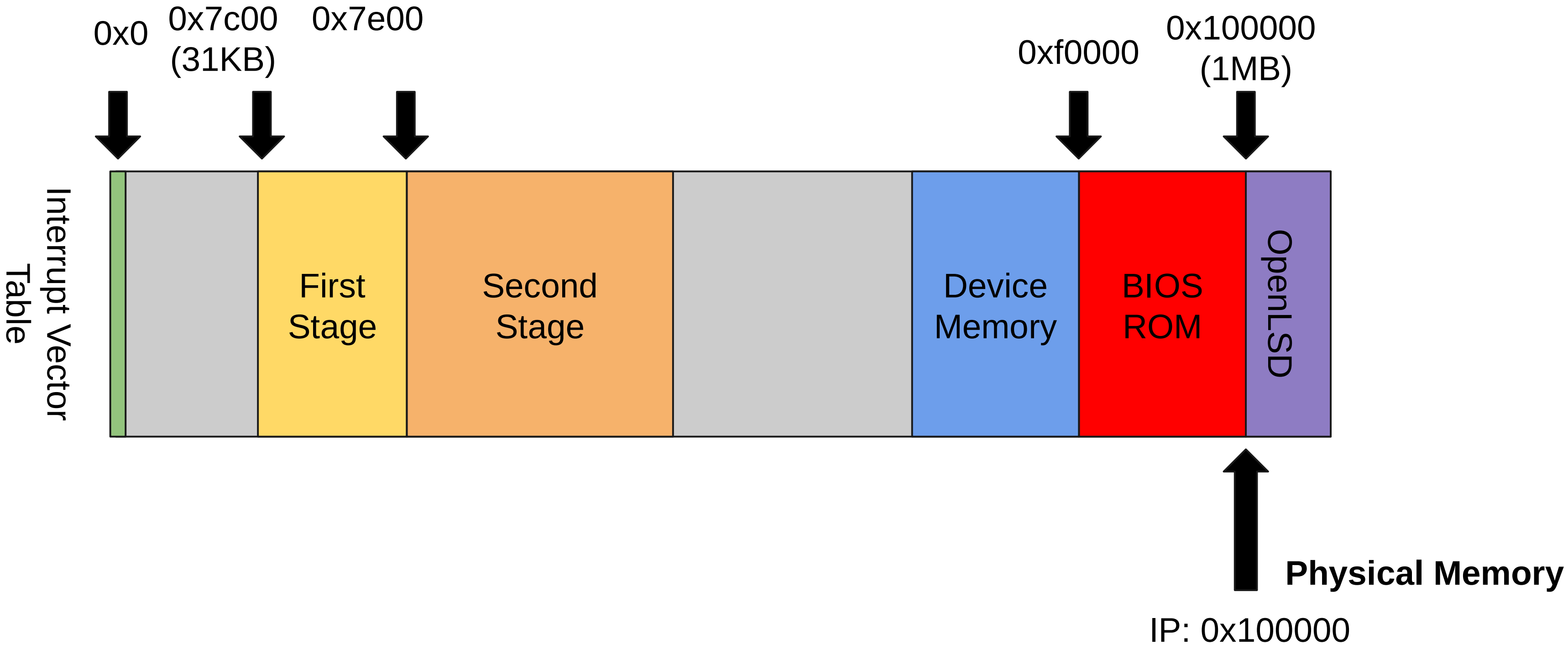
Physical Memory

OpenLSD: boot/main.c



Physical Memory

OpenLSD: kernel/boot.S



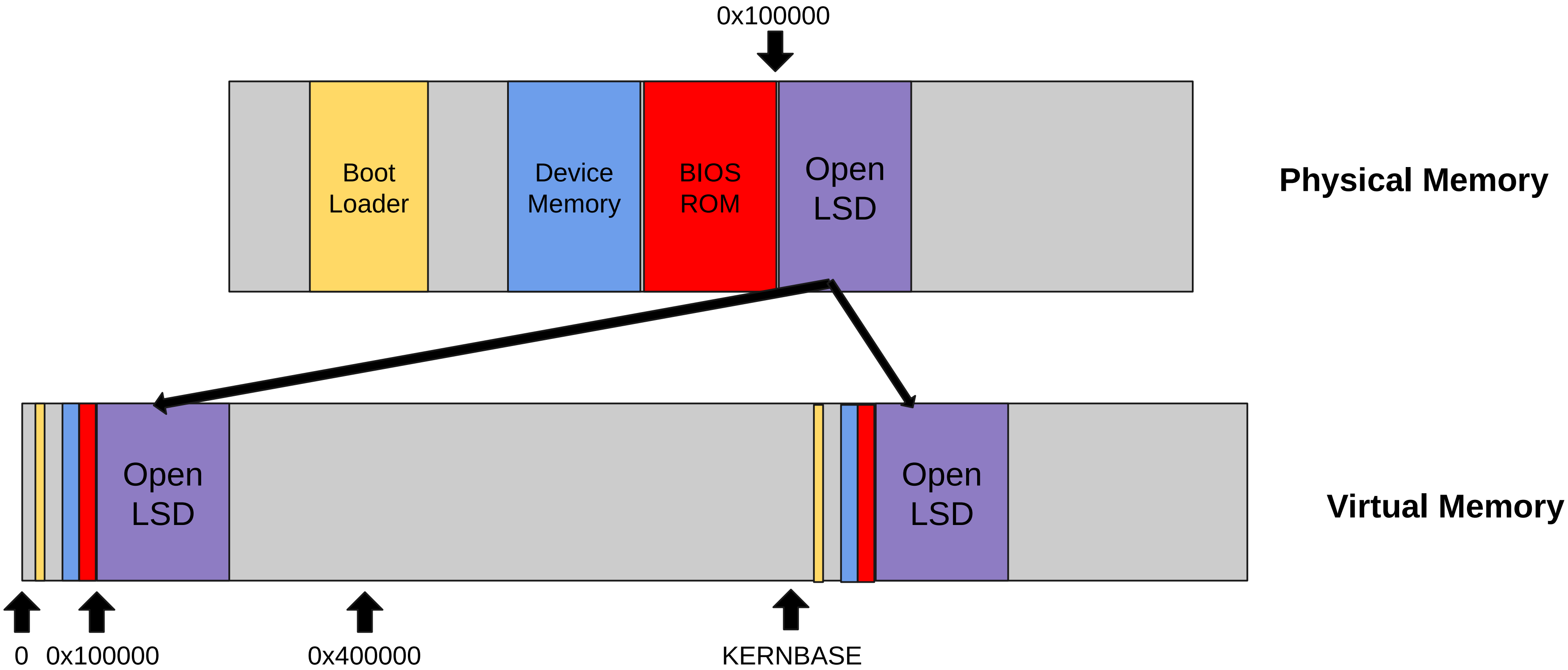
Entering the kernel

- Enable compatibility mode
- Enable paging
- Jump to long mode (64 bit mode)
- Initialize global kernel variables (BSS)
- Initialize console
- Initialize memory (lab 1)

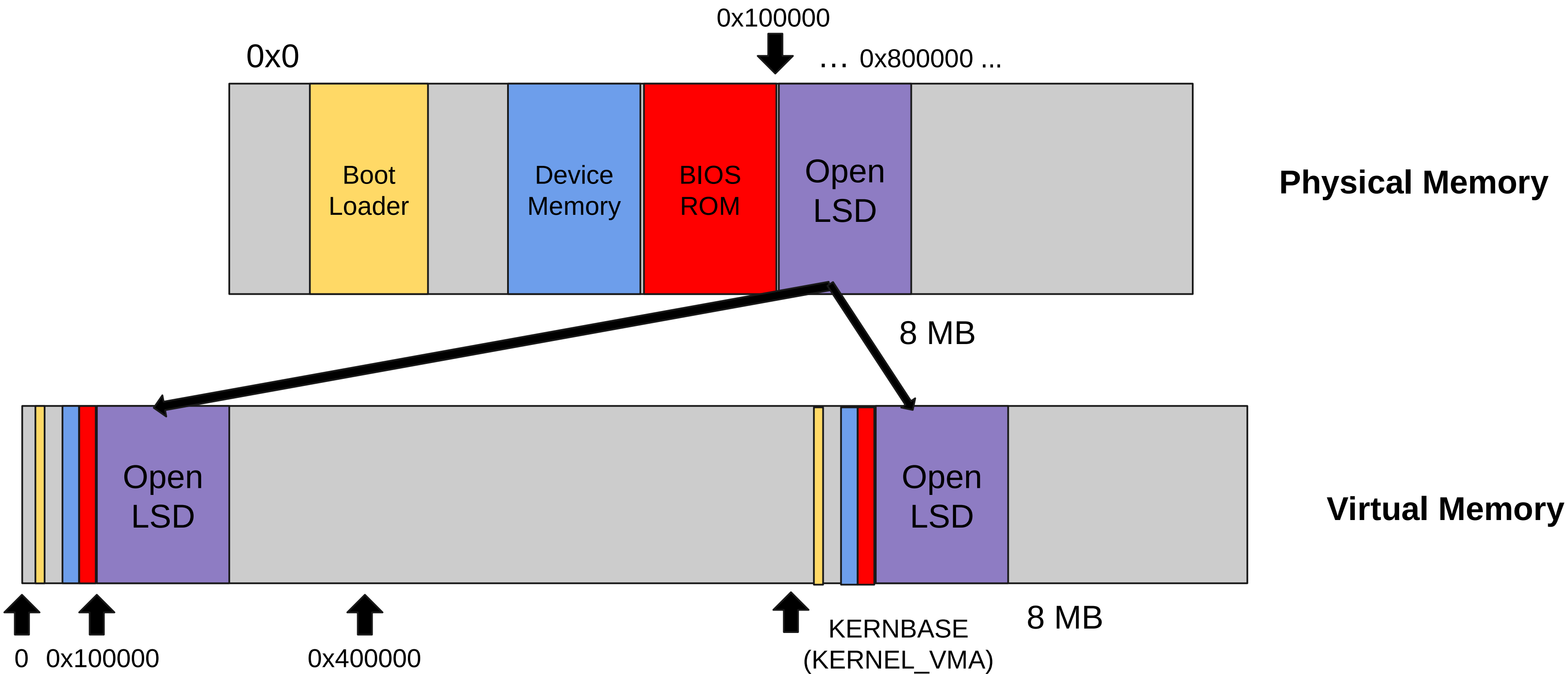
Mapping Virtual to Physical Addresses

- Need to translate virtual addresses to physical addresses
- Done by the CPU (MMU) through page tables
- We will discuss them in detail on Friday
- OpenLSD starts with a static “bootstrapping” page table

OpenLSD initial address space



OpenLSD initial address space



References

1. Booting a PC, https://sipb.mit.edu/iap/6.828/lab/lab1/
2. Bootstrapping, https://www.cs.columbia.edu/~junfeng/11sp-w4118/lectures/boot.pdf
3. Setting up long mode, http://wiki.osdev.org/Setting_Up_Long_Mode
4. How computers boot up, http://duartes.org/gustavo/blog/post/how-computers-boot-up/
5. The (Linux) kernel boot process, http://duartes.org/gustavo/blog/post/kernel-boot-process/
6. http://wiki.osdev.org/UEFI