

# Operating Systems Design

## 2. Booting

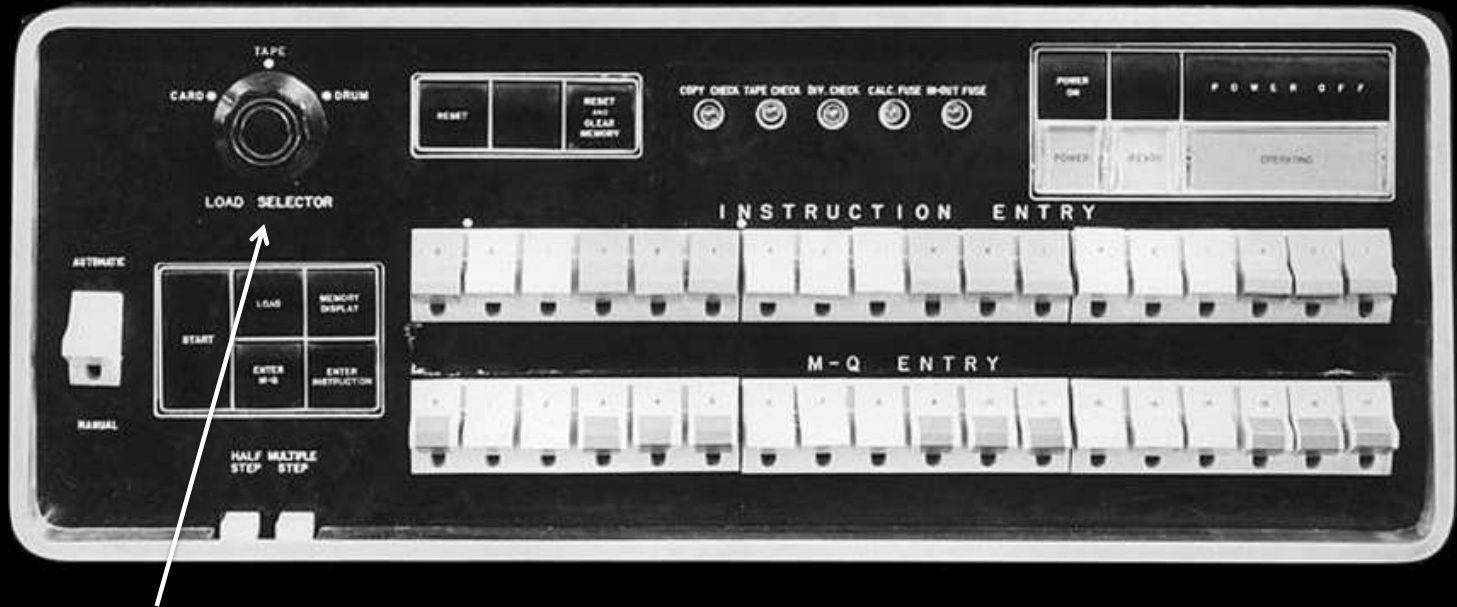
Paul Krzyzanowski  
pxk@cs.rutgers.edu

# What runs first?

---

- Boot loader
  - A program that loads a bigger program (e.g., the OS)

# Booting



Load selector: Card, Tape, Drum

<http://www.computer-history.info/Page4.dir/pages/IBM.701.dir/images/IBM701.jpg>

# Booting

GNU GRUB version 1.99~rc1

```
Ubuntu, with Linux 2.6.38-8-generic
Ubuntu, with Linux 2.6.38-8-generic (recovery mode)
Chainload to rEFIt
Chainload to ELILO
```

Use the ▲ and ▼ keys to select which entry is highlighted.  
Press enter to boot the selected OS, 'e' to edit the commands  
before booting or 'c' for a command-line.

# Multi-stage boot loader (chain loading)

---

- First stage boot loader

- Often primitive enough that an operator could enter the code via front panel switches ... or it could sit in the first block of a disk

- Second stage loader

- More sophisticated and included error checking

- Second stage loader may give the user a choice:

- Different operating systems
- Boot a test program
- Enable diagnostic modes (e.g., safe boot) in the OS

# Transfer of control

---

- When the boot loader finishes loading the OS, it transfers control to it
- The OS will initialize itself and load device drivers as needed

# Intel/AMD PC Startup

---

- CPU reset at startup
- Start execution at 0xfffffffff0
  - Jump instruction to BIOS code in non-volatile memory
    - Near the top of 32-bit addressable memory map
    - **Reset vector**: jump to firmware initialization code
  - Processor starts in **Real Mode**
    - 20-bit address space (top 12 address lines held high)
    - Direct access to I/O, interrupts, and memory

*Protected mode*

# BIOS

---

- BIOS = Basic Input/Output System
- Found in Intel-based 16- and 32-bit PCs
- Code resident in ROM or non-volatile flash memory
- Background: CP/M (MS-DOS was almost a clone)
  - Console Command Processor (CCP)
  - Basic Disk Operating System (BDOS)
  - Basic Input/Output System (BIOS): *all the device interfaces*



# PC Startup

---

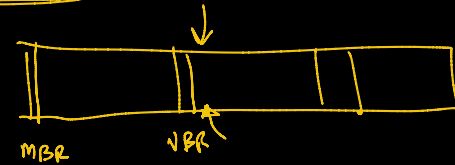
- BIOS executes:
  - Power-on self-test (POST)
  - Detect video card's BIOS – execute video initialization
  - Detect other device BIOS – initialize
  - Display start-up screen
  - Brief memory test
  - Set memory, drive parameters
  - Configure Plug & Play devices: PCIe, USB, SATA, SPI
    - Assign resources (DMA channels & IRQs)
  - Identify boot device:
    - Load block 0 (Master Boot Record) to 0x7c00 and jump there

# Booting Windows (XP/2003/2000/NT)

- The BIOS does not know file systems but can read disk blocks

- MBR = Master Boot Record = Block 0 of disk (512 bytes)

- Small boot loader (chain loader,  $\leq 440$  bytes)
- Disk signature (4 bytes)
- Disk partition table (16 bytes per partition \* 4)



- BIOS firmware loads and executes the contents of the MBR
- MBR code scans through partition table and loads the Volume Boot Record (VBR) for that partition
  - Identifies partition type & size
  - Contains Instruction Program Loader that executes startup code
  - IPL reads additional sectors to load NTLDR

# Booting other systems on a PC

---

- Example: GRUB (Grand Unified Boot Loader)
- MBR contains GRUB Stage 1
  - Or another boot loader that may boot GRUB Stage 1 from the Volume Boot Record
- Stage 1 loads Stage 2
  - Present user with choice of operating systems to boot
  - Optionally specify boot parameters
  - Load selected kernel and run the kernel
  - For Windows (which is not Multiboot compliant),
    - Run MBR code or Windows boot menu
    - Multiboot specification:
      - Free Software Foundation spec on loading multiple kernels using a single boot loader

# PCs and (U)EFI

---

- ~2005: Unified Extensible Firmware Interface (UEFI)
- Typically used for 32- and 64-bit architectures
  - Including Macs, which also have BIOS support for Windows
- Goal:
  - Create a successor to the BIOS
    - no restrictions on running in 16-bit 8086 mode with 20-bit addressing

# UEFI Includes

---

- Preserved from BIOS:
  - Power management (Advanced Configuration & Power Interface, ACPI)
  - System management components from the BIOS
- Support for larger disks
  - BIOS only supported 4 partitions per disk, each up to 2.2 TB per partition
  - EFI supports max partition size of 9.4 ZB ( $9.4 \times 10^{21}$  bytes)
- Pre-boot execution environment with direct access to all memory
- Device drivers, including the ability to interpret architecture-independent EFI Byte Code (EBC)
- Boot manager: lets you select and load an OS
  - *No need for a dedicated boot loader*
  - *Stick your files in the EFI boot partition and EFI can load them*
- Extensible: extensions can be loaded into non-volatile memory

# EFI Booting

---

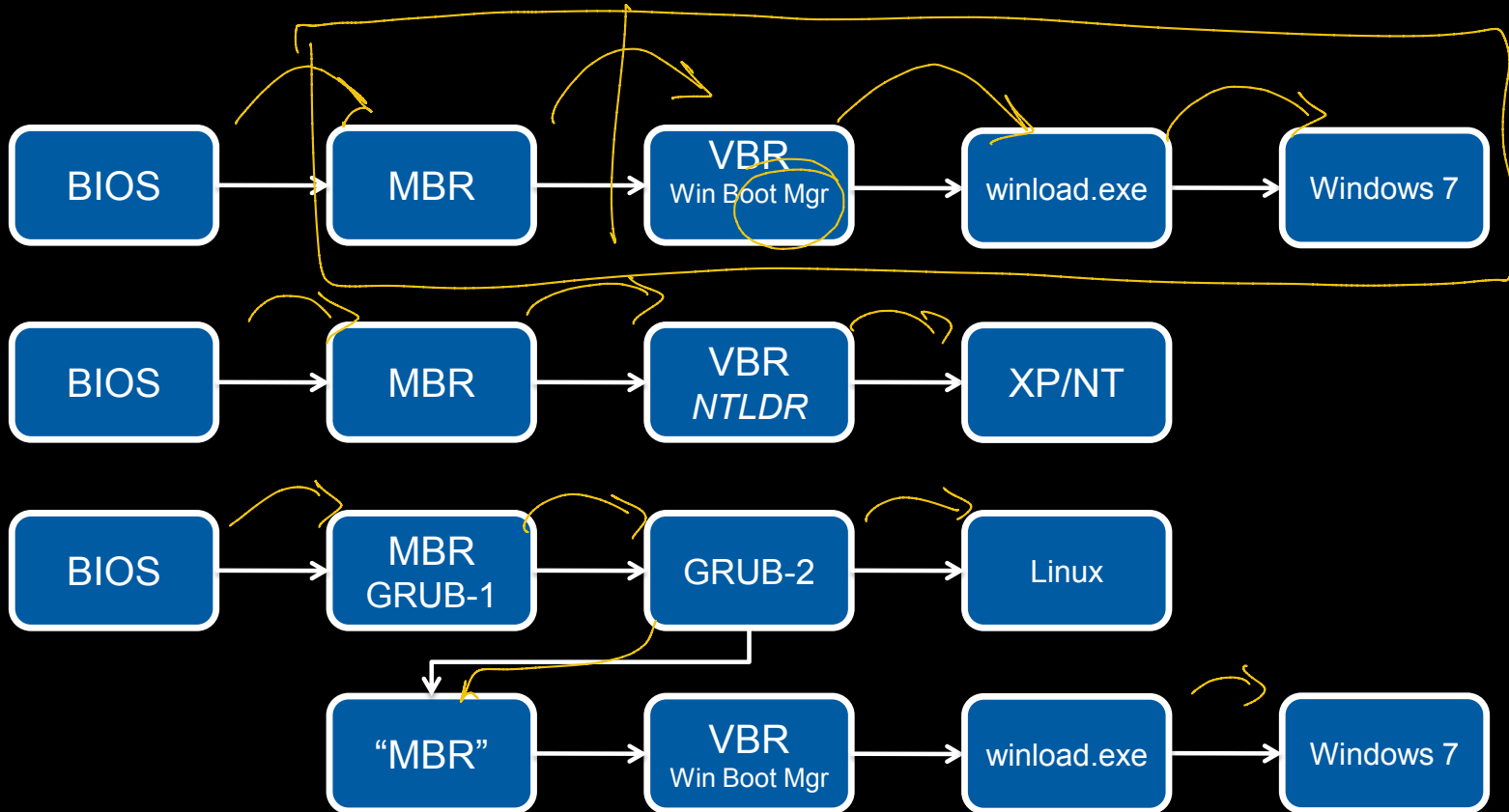
- No need for MBR code (*ignore block 0*)
- Read GUID Partition Table (GPT)
  - Describes layout of the partition table on a disk (blocks 1-33)
- EFI understands the FAT file system
  - Apple's EFI knows HFS+ in addition
- Read programs stored as *files* in the EFI System Partition:
  - Windows 7, Vista, Windows 2008 (64-bit Microsoft systems):
    - Windows Boot Manager (BOOTMGR) is in the EFI partition
  - NT (IA-64): IA64ldr
  - Linux: elilo.efi (ELILO = EFI Linux Boot Loader)
  - OS X: boot.efi

# Non-Intel Systems

---

- Power on: execute boot ROM code (typically NOR Flash)
  - Often embedded in the CPU ASIC
- Boot ROM code detects boot media
  - Loads first stage boot loader (sometimes to internal RAM)
  - Initialize RAM
  - Execute boot loader
- Second stage boot loader loads kernel into RAM
  - For Linux, typically GRUB for larger systems
  - uBoot for embedded systems
  - Set up network support, memory protection, security options

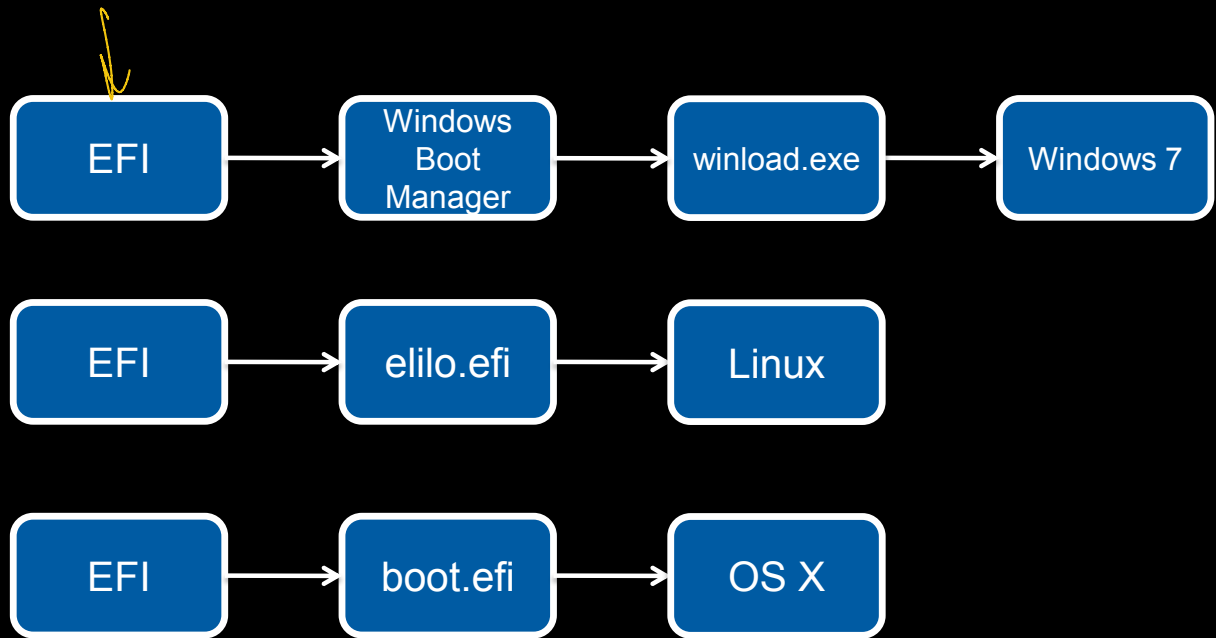
# Summary





# Summary

---



# The End