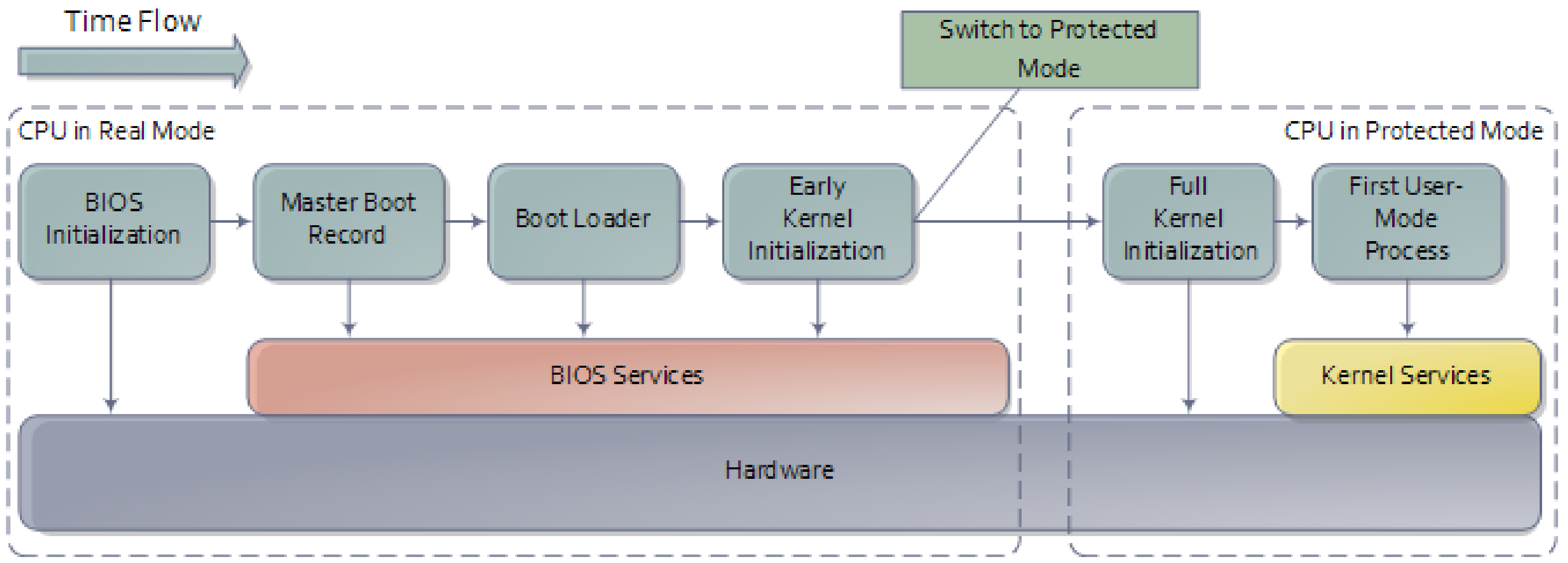
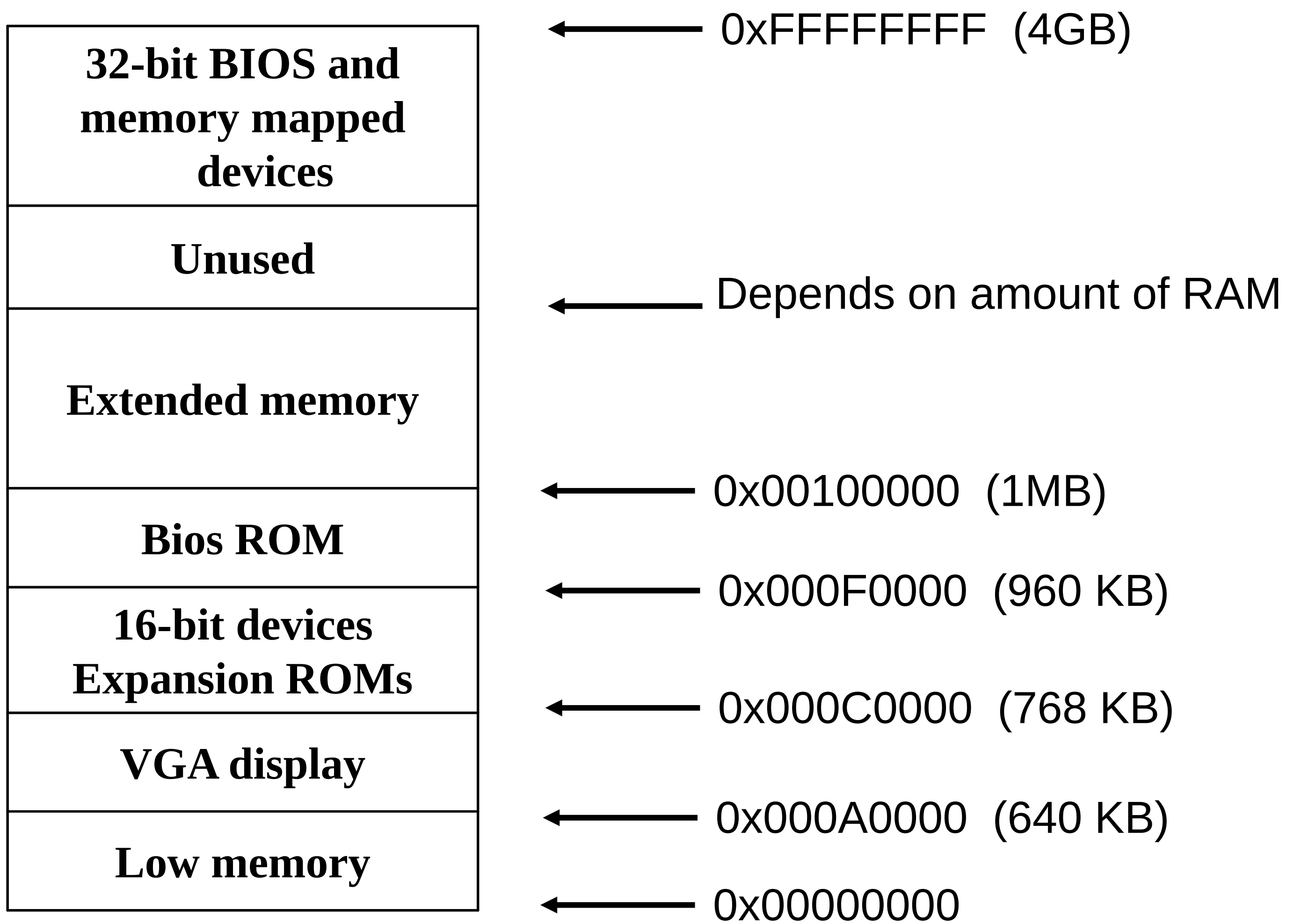
Booting a PC to run a kernel

after http://www.cs.ucla.edu/~/kohler/class/06f-aos/lab1.html

Booting steps

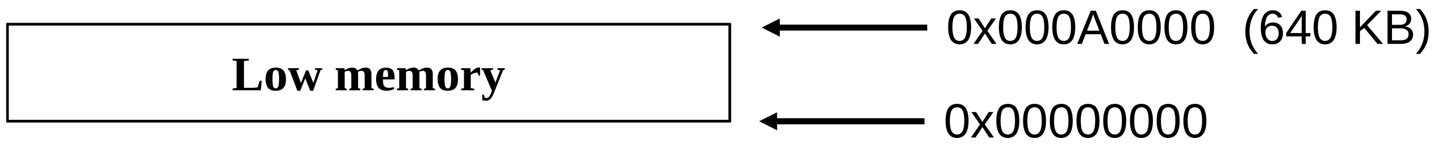


Booting a PC to run a kernel



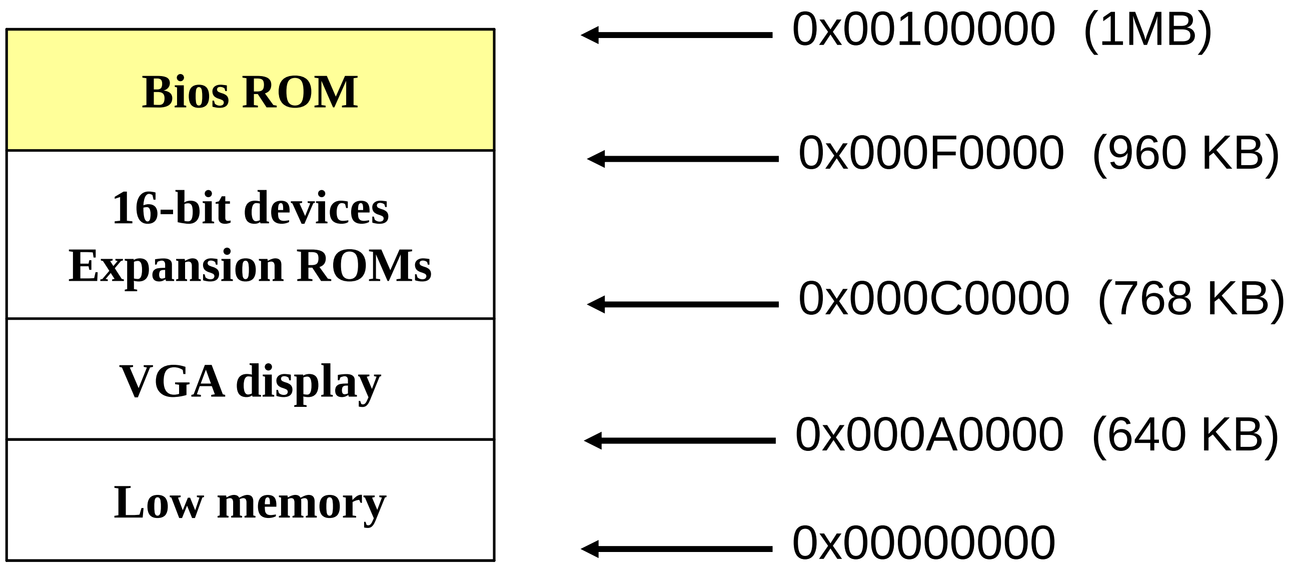
PC physical address space layout

- **The first PCs, which were based on the 16-bit Intel 8088 processor, were only capable of addressing 1MB of physical memory.**
- **The physical address space of an early PC would therefore start at 0x00000000 but end at 0x000FFFFF instead of 0xFFFFFFFF.**
- **The 640KB area marked "Low Memory" was the *only* random-access memory (RAM) that an early PC could use.**



PC physical address space layout

- **The 384KB area from 0x000A0000 through 0x000FFFFF was reserved by the hardware for special uses such as video display buffers and firmware held in nonvolatile memory.**
- **The most important part of this reserved area is the Basic Input/Output System (BIOS), which occupies the 64KB region from 0x000F0000 through 0x000FFFFF.**



The BIOS

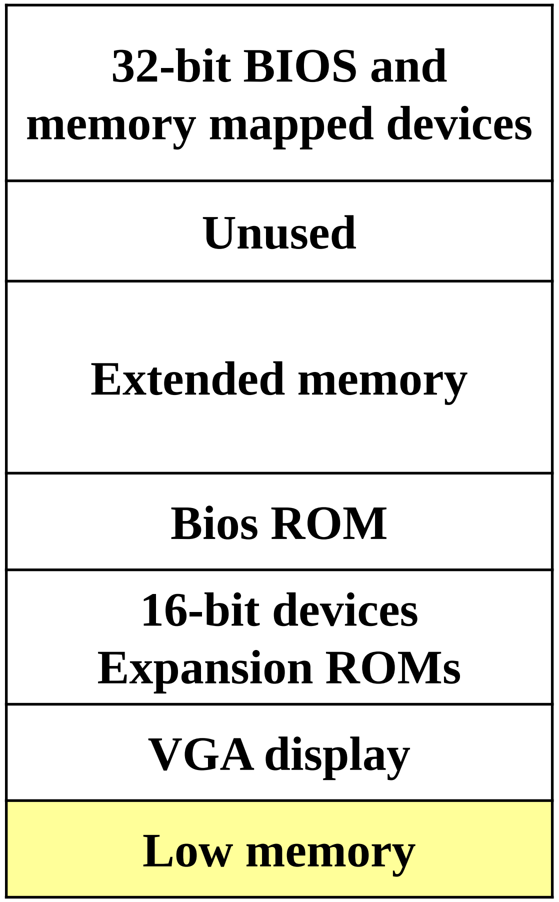
32-bit BIOS and memory mapped devices
Unused
Extended memory
Bios ROM
16-bit devices Expansion ROMs
VGA display
Low memory

- **More BIOS is located at the high end of the 32-bit address range for use by 32-bit PCI devices.**

The BIOS is responsible for performing Power-On-Self-Test and basic system initialization such as activating the video card and checking the amount of memory installed.

- **After performing this initialization, the BIOS loads the operating system from some appropriate location such as floppy disk, hard disk, CD-ROM, or the network, and passes control of the machine to the operating system**

PC physical address space layout


32-bit BIOS and memory mapped devices
Unused
Extended memory
Bios ROM
16-bit devices Expansion ROMs
VGA display
Low memory

- **Intel 80286 and 80386 processors supported 16MB and 4GB physical address spaces respectively, but the PC architects preserved the original layout for the low 1MB of physical address space for backward compatibility with existing software.**
- **Modern PCs therefore have a "hole" in physical memory from 0x000A0000 to 0x00100000, dividing RAM into "low" or "conventional memory" (the first 640KB) and "extended memory" (everything else).**

ROM BIOS in action

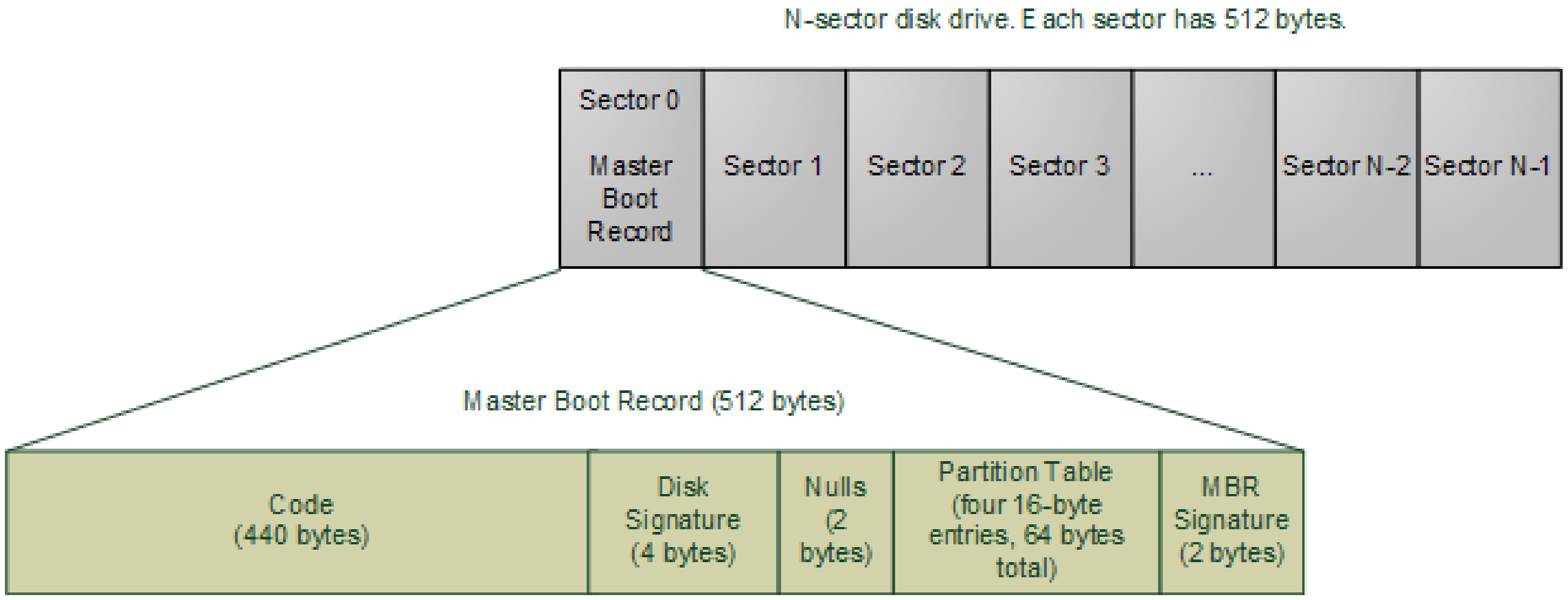
- 486 and later processors start executing at physical address 0xFFFFFFF0, which is at the very top of the memory space, in an area reserved for the ROM BIOS.
- The first instruction is: `jmp far f000:e05b`
- This instruction jumps to the normal BIOS, which is located in the 64KB region from 0xF0000 to 0xFFFFF mentioned above.
- The CPU starts in real mode, so the `jmp far` instruction is a real mode jump that restores us to low memory.
- In real mode, the *segmented address* segment:offset translates to the physical address segment*16+offset
- Thus, f000:e05b translates to 0x000fe05b

```
0x0000f000 * 16 =  0x000f0000 +
                   0x0000e05b =
                   0x000fe05b
```

ROM BIOS in action

- No GDT, LDT or paging table is needed by the CPU in real mode
- The code that initializes these data structures must run in real mode
- When the BIOS runs
  - it initializes the PCI bus and all the important devices it knows about, (in particular VGA display)
  - then it searches for a bootable device such as a floppy, hard drive, or CD-ROM.
  - when it finds a bootable disk, it reads the boot loader from the disk and transfers control to it.

Master Boot Record


N-sector disk drive. Each sector has 512 bytes.
Sector 0 Master Boot Record
Sector 1
Sector 2
Sector 3
...
Sector N-2
Sector N-1
Master Boot Record (512 bytes)
Code (440 bytes)
Disk Signature (4 bytes)
Nulls (2 bytes)
Partition Table (four 16-byte entries, 64 bytes total)
MBR Signature (2 bytes)

- The BIOS loads the contents of the MBR into memory location 0x7c00 and jumps to that location to start executing whatever code is in the MBR.

The Boot Loader

Is the program run by the BIOS to load the image of a kernel into RAM.

- **Floppy and hard disks for PCs are by historical convention divided up into 512 byte regions called *sectors*.**
- **If the disk is bootable, the first sector is called the *boot sector*, since this is where the Boot Loader code resides.**
- **When the BIOS finds a bootable floppy or hard disk, it loads the 512-byte boot sector into low memory, at physical addresses**

`0x7c00` through `0x7dff`

$$\texttt{0x7e00 - 0x7c00 = 0x0200 = } 2^9 \texttt{ = 512}$$

- **and then uses a `jmp` instruction to set the `CS:IP to 0000:7c00`, passing control to the Boot Loader.**

BIOS and GRUB

- The BIOS loads the contents of the MBR into memory location 0x7c00 and jumps to that location to start executing the code in the MBR.
- The MBR itself contains the first stage of the boot loader. GRUB calls this stage 1.
  - The code in the MBR loads another sector from disk that contains additional bootstrap code.
  - This sector might be the boot sector for a partition, but could also be a sector that was hard-coded into the MBR code when the MBR was installed.
- Code loaded in step 2 then read a file containing the second stage of the boot loader. GRUB calls this stage 2.
  - The stage 2 code reads a boot configuration file (grub.conf in GRUB). It then presents boot choices to the user or simply goes ahead in a single-boot system.
- Then the boot loader code needs to start a kernel. It must know enough about file systems to read the kernel from the boot partition.

Installing GRUB on a hard disk image file

- Installing GRUB on a HD image file
- Creating a skeleton kernel
- Adding the kernel image to the HD image file
- Debugging GRUB and the kernel

To be done in Laboratory