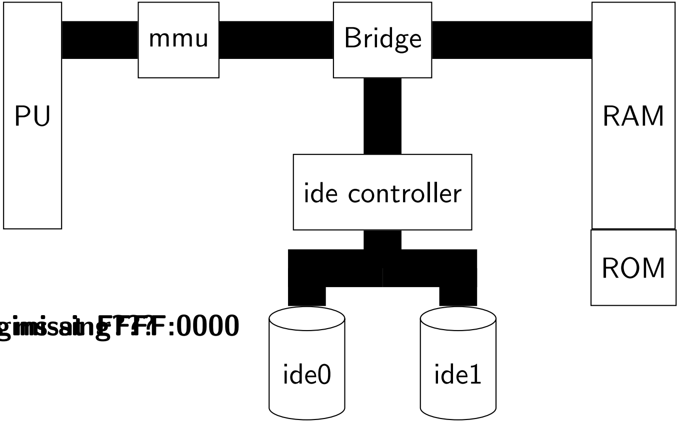
xv6©-rev10
(Copyright Frans Kaashoek, Robert Morris, and Russ Cox.)
The Boot Sequence

Carmi Merimovich

Tel-Aviv Academic College

March 10, 2017

x86 Power on



Processor begins at FFFF:0000

What's missing???

Knowledge needed for general booting

- ROM structure (and contents).
- RAM structure.
- ELF structure.
- File system structure.

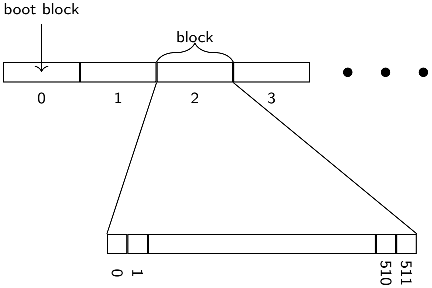
This is way too much knowledge at this point.

- We will use simplfying assumptions.
- (And) we will not see the code now.

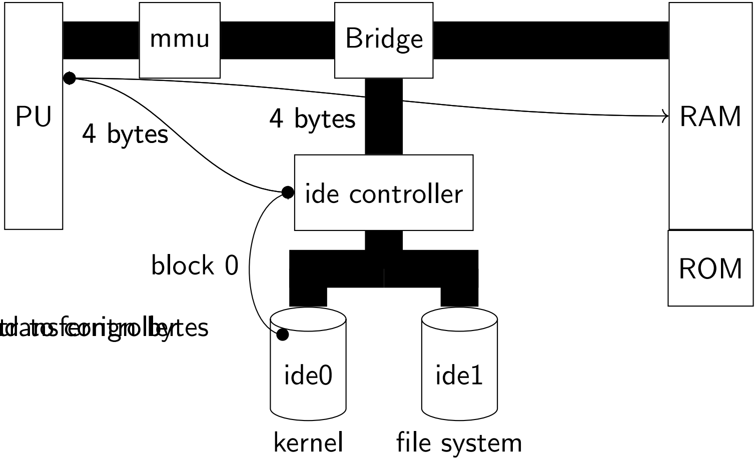
ROM and the Boot Block

- Should ROM code load the kernel into RAM?
  - No.
- ROM code loads the **boot block** into RAM.
  - ROM might scan several devices in order to find one with a boot block.
- The boot block is kernel dependent.
- The boot block loads the kernel code, if possible.
  - (elective) Ignoring Lilo, Grub et at.
- Usually the kernel code is a standard executable file.
- Thus the boot block code should know:
  - File system structure.
  - Executable file structure.

Logical disk structure



Boot Block Loading by the ROM



Birds eye view on getting the kernel

1. POWERUP.
   - Primary processor executes instructions. MMU inactive.
2. ROM code loades 512-bytes boot block to address 0x07C0 and up.
   - ROM terminates by JMPing to address 0x07C0.
3. Boot block code loads the kernel into RAM.
   - Boot block code terminates by JMPing into the kernel's entry point.
   - The entry point address is found at a fixed location in the kernel ELF.
   - Then entry point code is in Assembly and is labeled **entry**.
4. **entry** sets up a temporary kernel programming model.
   - MMU is activated with a table coding the following translation:

$$[\mathtt{0x80000000}, \mathtt{0x803FFFFF}] \mapsto [\mathtt{0x00000000}, \mathtt{0x003FFFFF}]$$

   - **esp** is set to the end of a 4KB buffer.
   - **entry** finishes by JMPing into `main`.

x86 ROM code

- Quite complicated, (i.e., BIOS).
- We need one crucial fact:
  - The ROM loads block 0 of the boot device begining with address 0000:07C0.
  - Then the ROM jumps into 0000:07C0.

xv6 Boot block operation

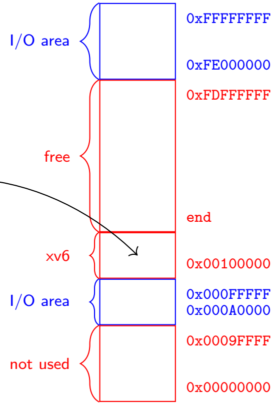
- The boot block loads the xv6 kernel into the RAM.
- It is not feasible in 512 bytes to have code which:
  - Understands ELF structure.
  - Understands the xv6 file system.
  - Puts the kernel code in arbitrary place in RAM.

x86 RAM structure

- Can be quite compilcated.
- **Simplification:** Assume there is RAM at the following addresses:
  - [0x00000000,0x000A0000).
  - [0x00100000,0x0E000000).
- **Simplification:** Load the kernel from address 0x00100000.

xv6 physical memory state after loading

boot block loads xv6



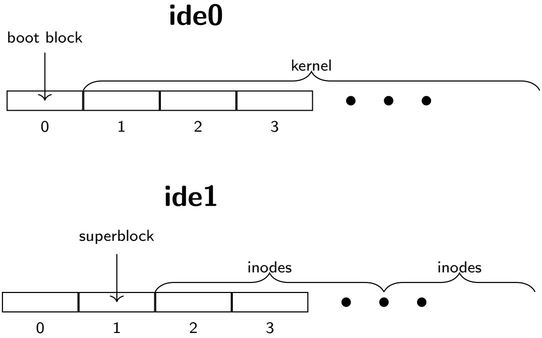
ELF structure

- We have to know the structure of a static ELF.
- We **will** study static ELF structure later in the course.
- Currenly, we will **ignore** the ELF loading code.

File system

- This will be really complicated at this point in time.
- For simplicity, xv6 uses a whole disk (ide0) to hold the kernel.
  - The kernel ELF file begins at block 1 of ide0.
- The real file system, with the OS utilities will reside on ide1.

"disk"s generated by the **makefile**



xv6.img and fs.img

```
xv6.img: bootblock kernel
 dd if=/dev/zero of=xv6.img count=10000
 dd if=bootblock of=xv6.img conv=notrunc
 dd if=kernel of=xv6.img seek=1 conv=notrunc

fs.img: mkfs README $(UPROGS)
 ./mkfs fs.img README $(UPROGS)
```