Week 12.b
CS 5600
03/29 2023

0. Last time
1. Disks
2. SSDs
3. Intro to file systems

CPU-I/O interactions, four approaches:
  * port-mapped I/O → keyboard, set cursor
  * memory-mapped I/O → Console putc
  * interrupts
  * via memory → DMA

PMIO / MMIO

- {polling, interrupt} X {Programmed IO, DMA}
- drivers.

Q: NVIDIA GPU driver → AMD GPU?

Q: NVIDIA GPU windows driver → Linux?

- Disk

less expensive (more durable?)

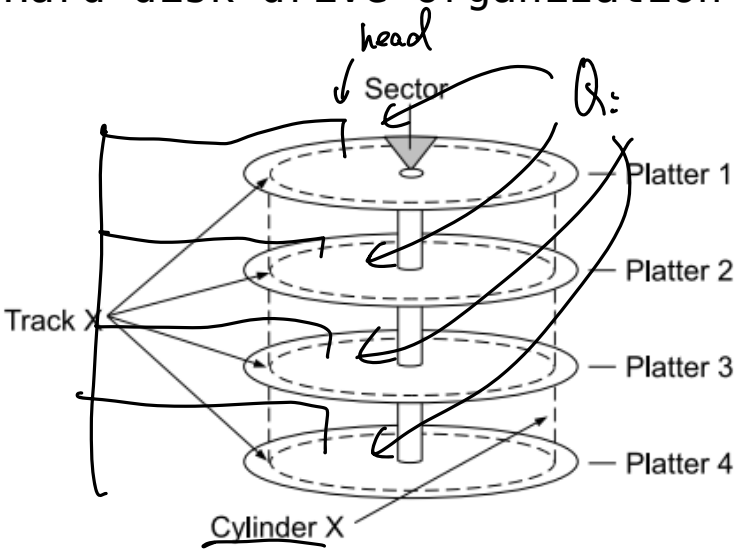
Persistent Memory

performant

tape → HDD → SSD → PM // DRAM → CPU Cache → Registers

Persistency ← → Volatile

Hard disk drive organization



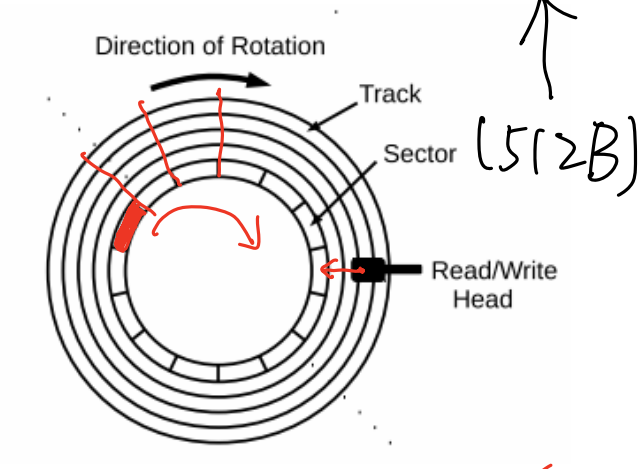


Fig 5.11, 5.12 from
http://www.ccs.neu.edu/~pjd/x600-book-v0903.pdf

A Flash-based SSD (logical diagram)

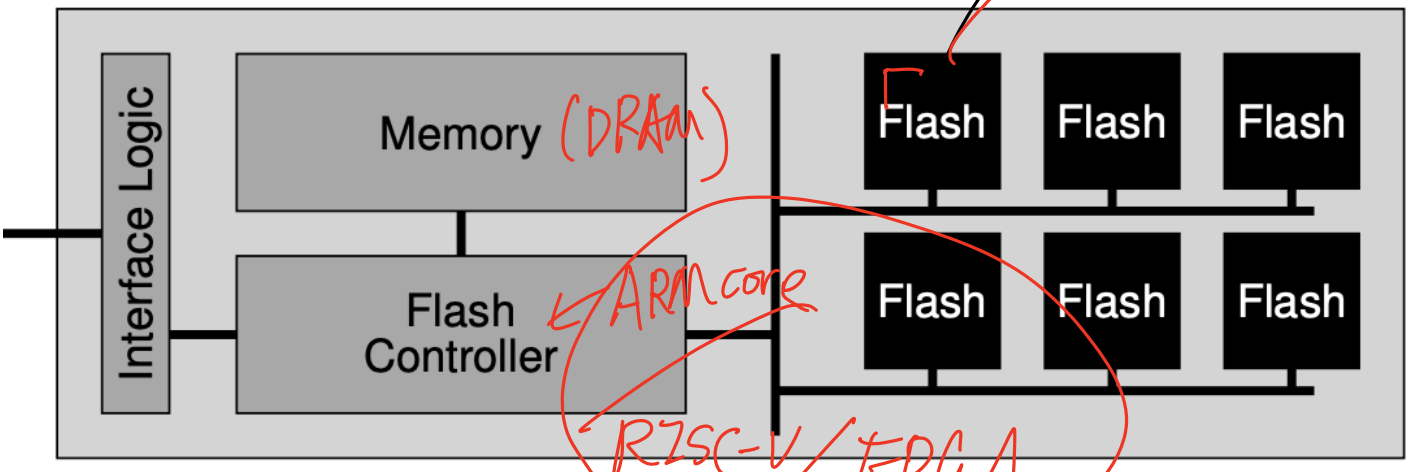
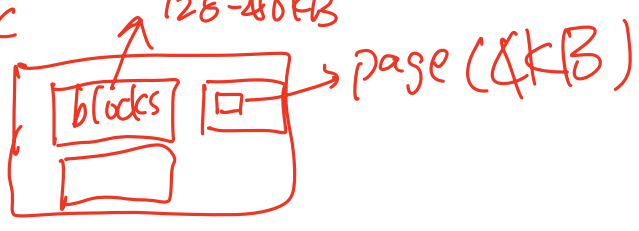


Fig 44.3 from
https://pages.cs.wisc.edu/~remzi/OSTEP/file-ssd.pdf



Common #s and performance

* capacity: Several TBs, (→ 20TBs)
* platters: 8
* number of cylinders: ~100,000s
* sectors per track: ~1000
* RPM: 5400 → 15000
* transfer rate: 50 ~ 200 MB/s
* mean time between failures: 1s, 0.01s, 1min, 2weeks, 1month, 5 yrs, ∞, 7yrs, 11yrs, 20yrs, 50yrs,

[question]

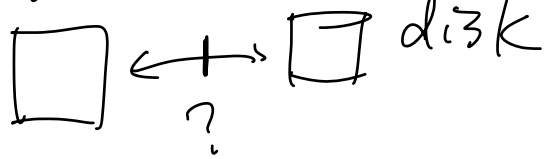
~1M hrs

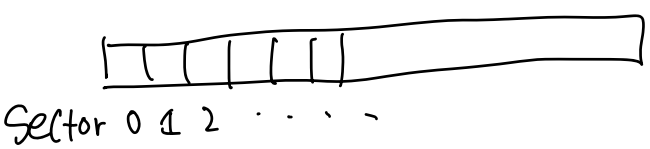
↳ 100yrs.

25,233 hrs.

< 3yrs.

driver





```
1   CS5600, Handout Week12.a
2
3   //   Code snippets borrowed from WeensyOS boot loader
4   //   They illustrate how kernel "talks" to a disk through programmed I/O:
5   //   the bootloader reads in the kernel from the disk.
6   //
7   //   See the functions boot_waitdisk() and boot_readsect(). Compare to
8   //   Figures 36.5 and 36.6 in OSTEP.
9
10
11  //   WeensyOS boot loader loads the kernel at address 0x40000 from
12  //   the first IDE hard disk.
13  //
14  //   A BOOT LOADER is a tiny program that loads an operating system into
15  //   memory. It has to be tiny because it can contain no more than 510 bytes
16  //   of instructions: it is stored in the disk's first 512-byte sector.
17
18
19      #define SECTORSIZE      512
20
21      // boot_readsect(dst, src_sect)
22      //    Read disk sector number `src_sect` into address `dst`.
23      static void boot_readsect(uintptr_t dst, uint32_t src_sect) {
24          // programmed I/O for "read sector"
25          boot_waitdisk();
26          outb(0x1F2, 1);             // send `count = 1` as an ATA argument
27          outb(0x1F3, src_sect);      // send `src_sect`, the sector number
28          outb(0x1F4, src_sect >> 8);
29          outb(0x1F5, src_sect >> 16);
30          outb(0x1F6, (src_sect >> 24) | 0xE0);
31          outb(0x1F7, 0x20);          // send the command: 0x20 = read sectors
32
33          // then move the data into memory
34          boot_waitdisk();
35          insl(0x1F0, (void*) dst, SECTORSIZE/4); // read 128 words from the disk
36      }
37
38
39      // boot_waitdisk
40      //    Wait for the disk to be ready.
41      static void boot_waitdisk(void) {
42          // Wait until the ATA status register says ready (0x40 is on)
43          // & not busy (0x80 is off)
44          while ((inb(0x1F7) & 0xC0) != 0x40) {
45              /* do nothing */
46          }
47      }
```

Page 1 of 1

BUMMER : wear-out

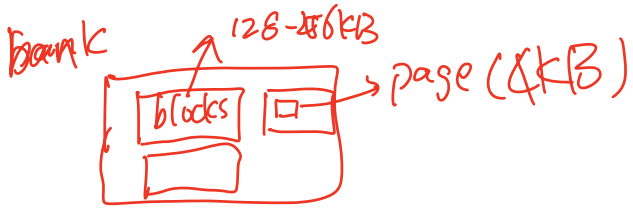
SSD

1 page → #erase ↗ 10,000

Ops:

- read : 1 page
- erase : 1 block, reset ALL bits to 1
- program: 1 page, set some bits to 0

(Cannot program a page twice w/o erasing)



Q: how to update a page?

at most 1