Disk I/O and File System

Agenda

- Disk
  - SD card driver
  - memory-mapped I/O
- From disk to file system
  - one-to-many virtualization
  - virtual block store and inodes
  - reading and writing a virtual block store

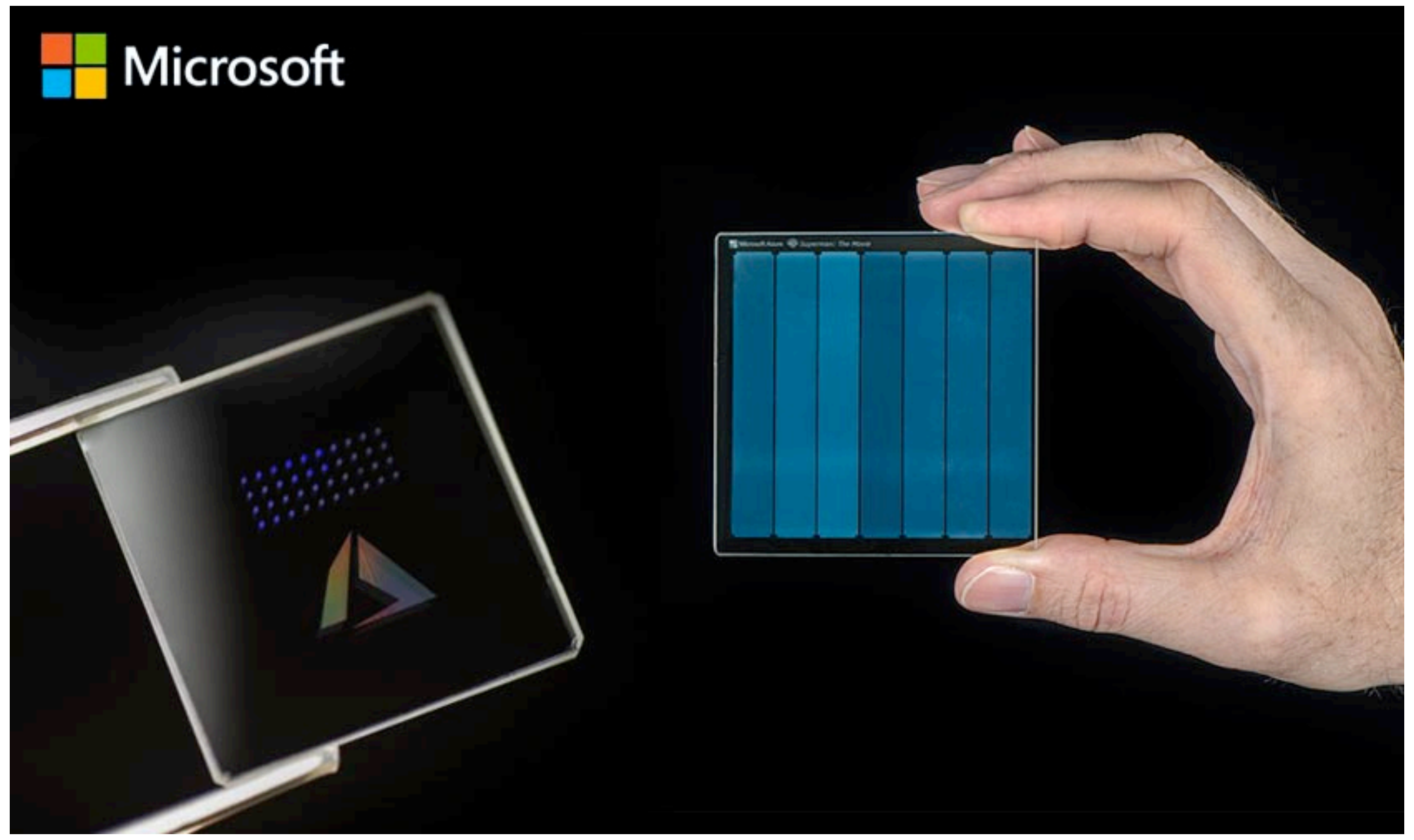
Disk: a sequence of blocks

Content	1st **block**	2nd **block**	3rd **block**	……	2^28th **block**
Address	0	1	2	……	2^28-1

- A block is usually 512 bytes
- 2^28 * 512 bytes → 2^37 bytes → 128 GB

From abstraction to implementation

Content	1st block	2nd block	3rd block	……	2^28th block
Address	0	1	2	……	2^28-1



Agenda

- Disk
  - ➡ SD card driver
  - memory-mapped I/O
- From disk to file system
  - one-to-many virtualization
  - virtual block store and inodes
  - reading and writing a virtual block store

Send a byte to SD card

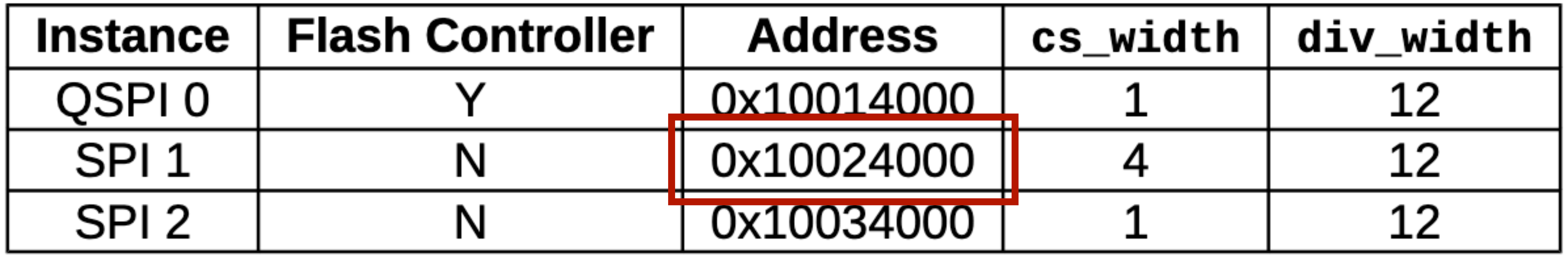
```
char send_data_byte (char byte) {
    /* Send the byte */
    while ((*(int*)(0x10024048)) & (1 << 31));
    (*(int*)(0x10024048)) = byte;

    /* Every byte sent will have one byte response */
    long rxdata;
    while ((rxdata = (*(int*)(0x1002404C))) & (1 << 31));
    return (char)(rxdata & 0xFF);
}
```

Receive a byte from SD card

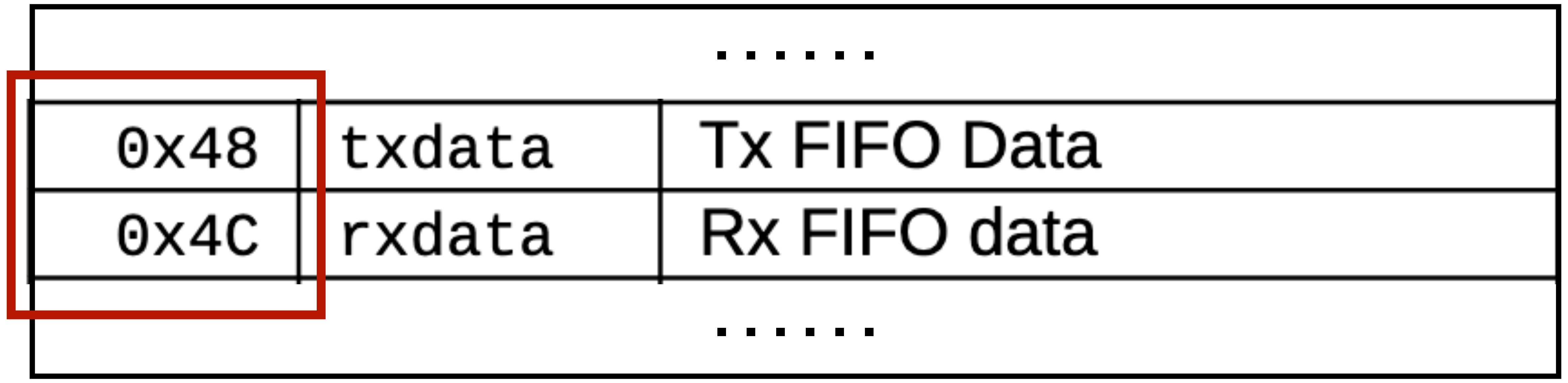
```
char recv_data_byte() {
    /* Send a dummy byte and get the response */
    return send_data_byte(0xFF);
}
```

Why 0x10024048 and 0x1002404C?



Instance	Flash Controller	Address	cs_width	div_width
QSPI 0	Y	0x10014000	1	12
SPI 1	N	0x10024000	4	12
SPI 2	N	0x10034000	1	12

Table 64: SPI Instances



	......	
0x48	txdata	Tx FIFO Data
0x4C	rxdata	Rx FIFO data
	......	

Table 65: Register offsets within the SPI memory map. Registers marked * are present only on controllers with the direct-map flash interface.

Chapter 19 of Sifive FE310 manual, v19p04
https://github.com/yhzhang0128/egos-2000/blob/main/references/sifive-fe310-v19p04.pdf

Read a block from SD card

```
/* Send a command to SD card reading block #128 */
int block_no = 128;
char *arg = (void*)&block_no;
char cmd17[] = {0x51, arg[3], arg[2], arg[1], arg[0], 0xFF};
for (int i = 0; i < 6; i++) send_data_byte(cmd17[i]);

/* Wait and receive 512 bytes */
while (recv_data_byte() != 0xFE);
for (int i = 0; i < 512; i++) dst[i] = recv_data_byte();
```

Agenda

- Disk
  - SD card driver
  - ➜ memory-mapped I/O
- From disk to file system
  - one-to-many virtualization
  - virtual block store and inodes
  - reading and writing a virtual block store

Take-away

Memory-mapped I/O: communicate with I/O devices using memory `load/store`

e.g., the 0x10024048 and 0x1002404C just mentioned

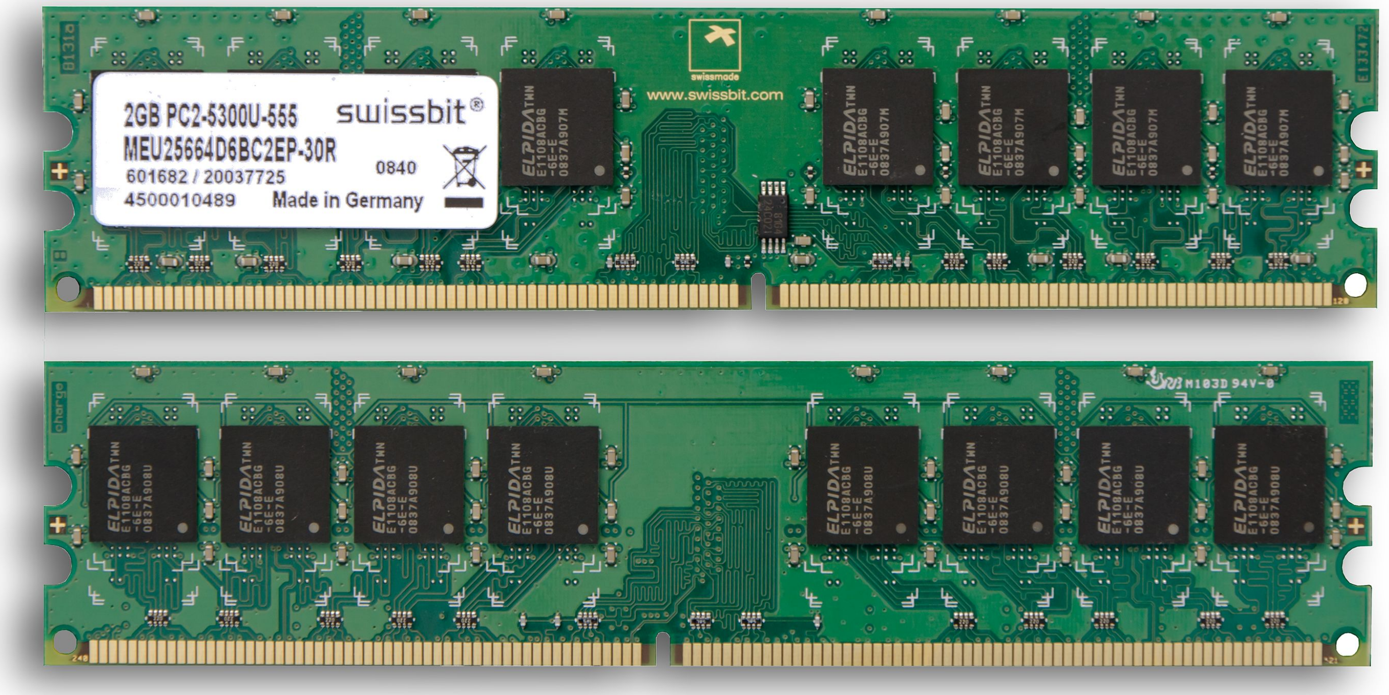
Brief history of Input/Output

- Port I/O
  - In Intel x86, there are special in/out instructions for I/O
- Memory-mapped I/O
  - In Intel x86 and RISC-V, there is an I/O hole in memory
    - read/write to I/O hole will not modify memory
    - instead, send/receive bytes to/from I/O devices

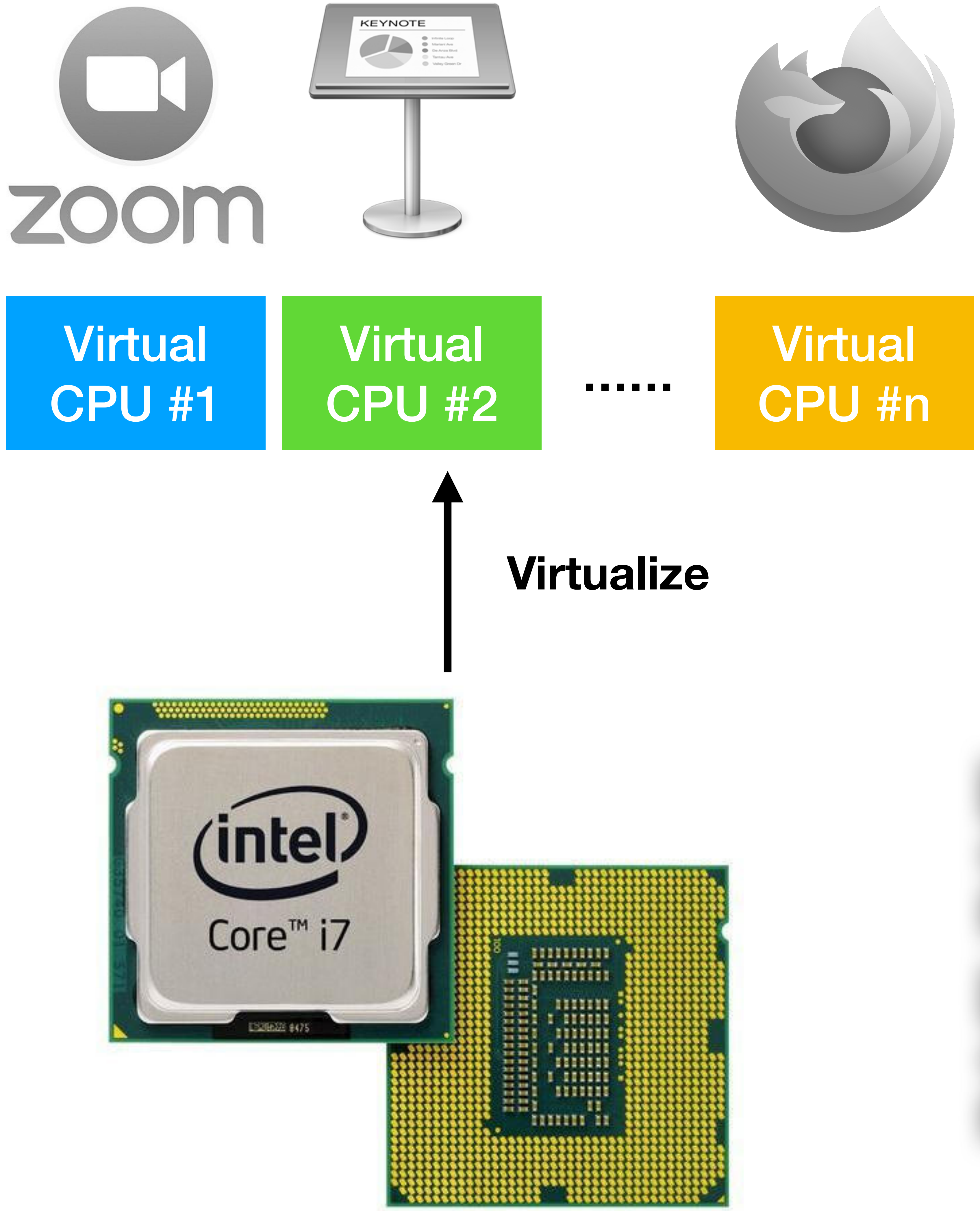
Agenda

- Disk
  - SD card driver
  - memory-mapped I/O
- From disk to file system
  - ➡ one-to-many virtualization
  - virtual block store and inodes
  - reading and writing a virtual block store

Recap: A computer has 3 key pieces



Scheduler is virtualizing the CPU



Virtual CPU #1

Virtual CPU #2

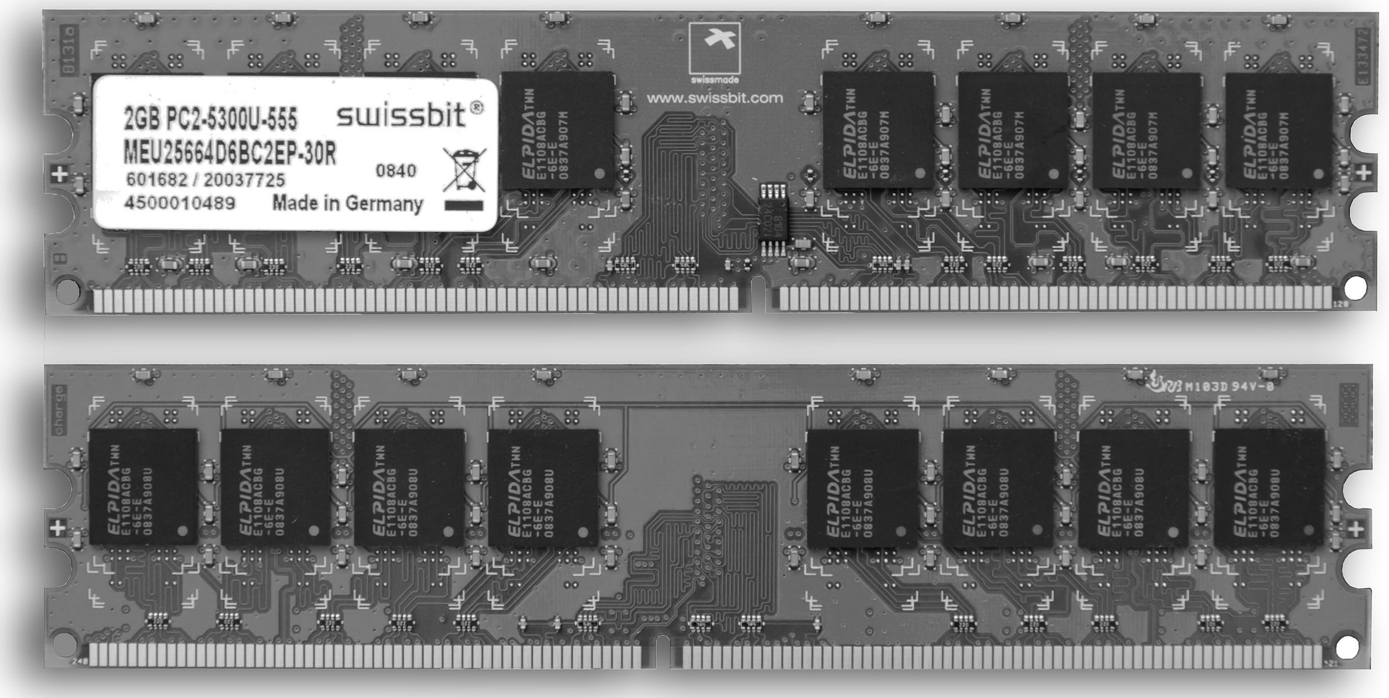
......

Virtual CPU #n

Virtualize

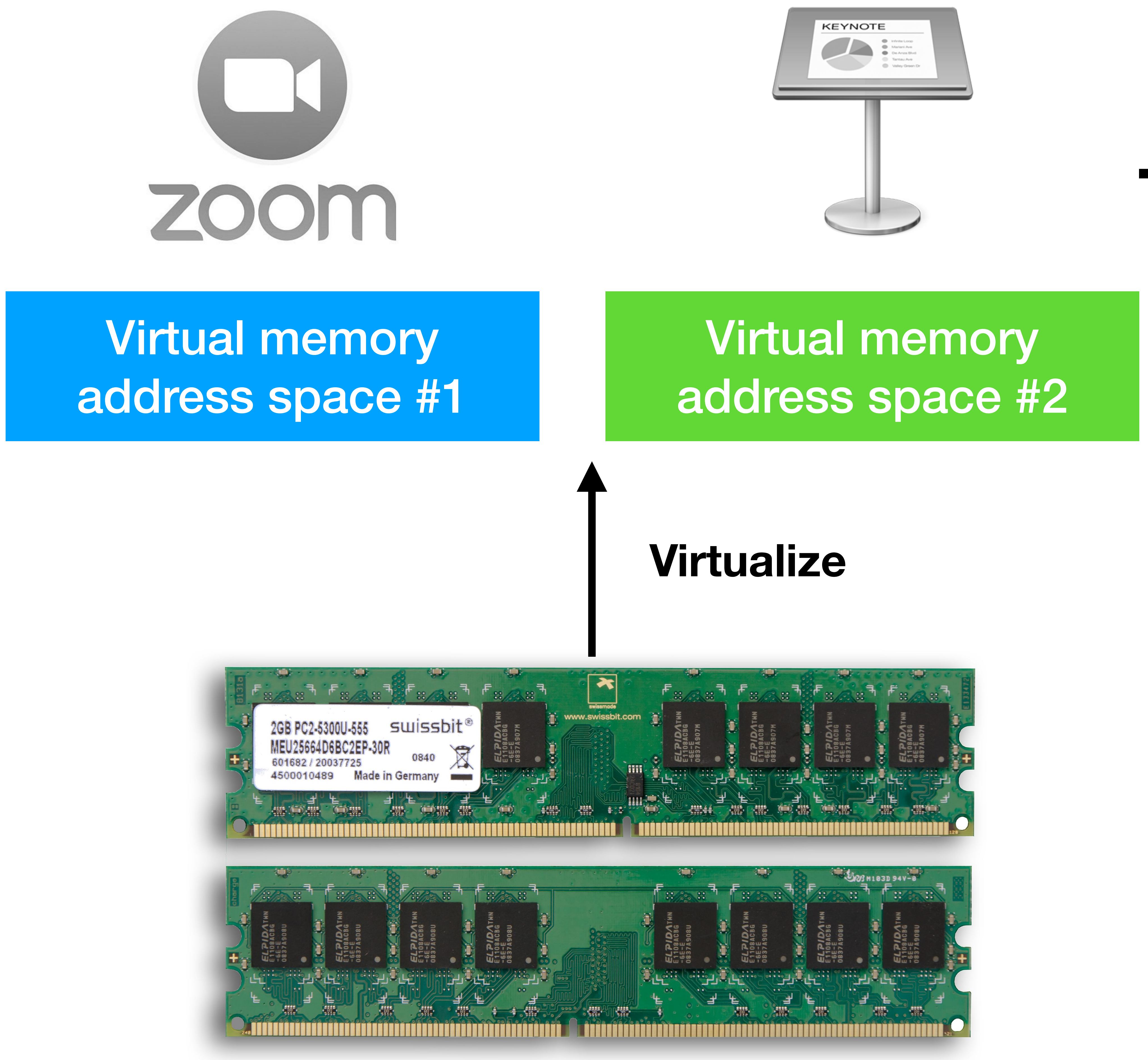
one physical CPU

→ many virtual CPUs



Virtual Memory

one physical memory



Virtual memory address space #1

Virtual memory address space #2

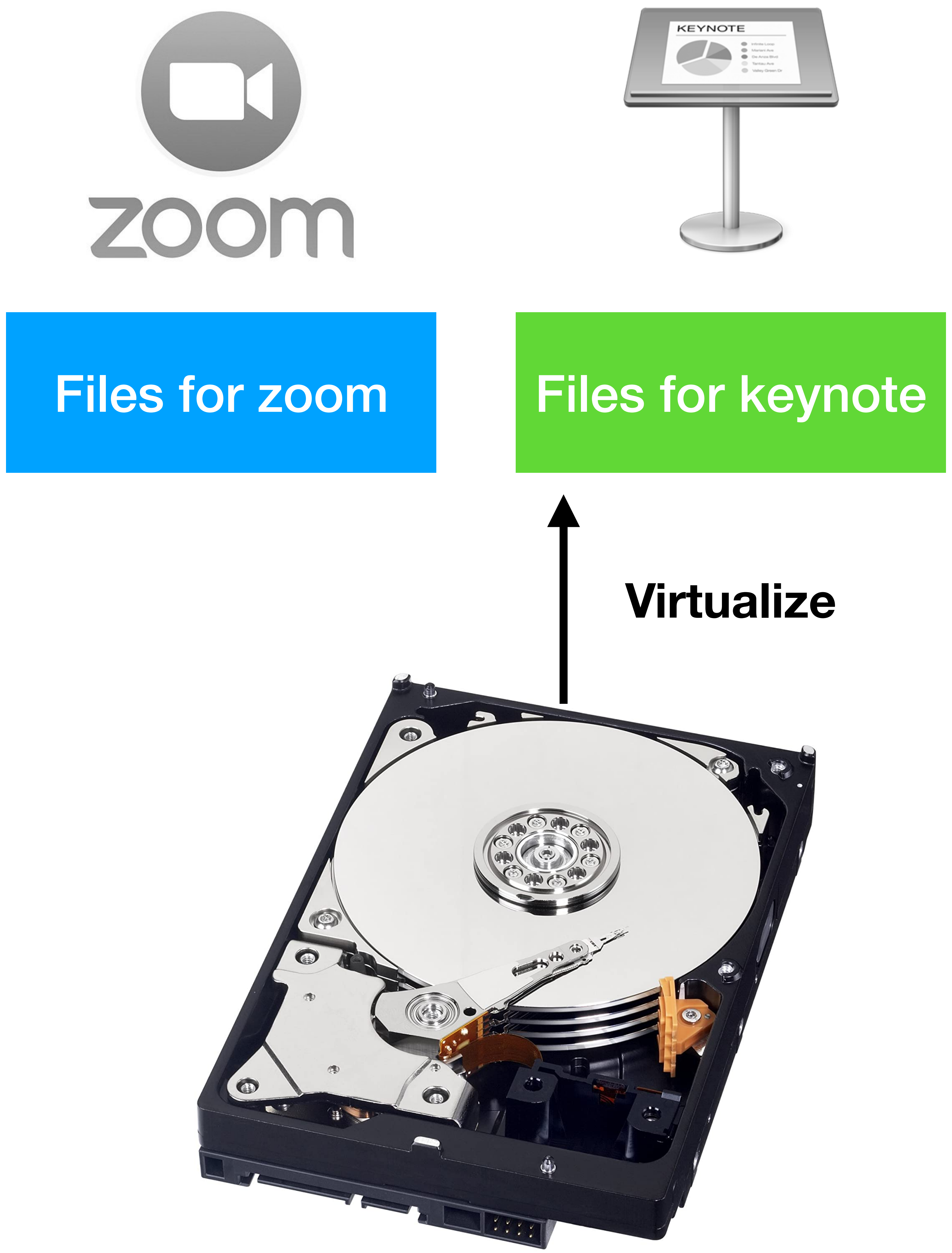
Virtualize

→ many virtual memory

File system is virtualizing the Disk

one physical disk

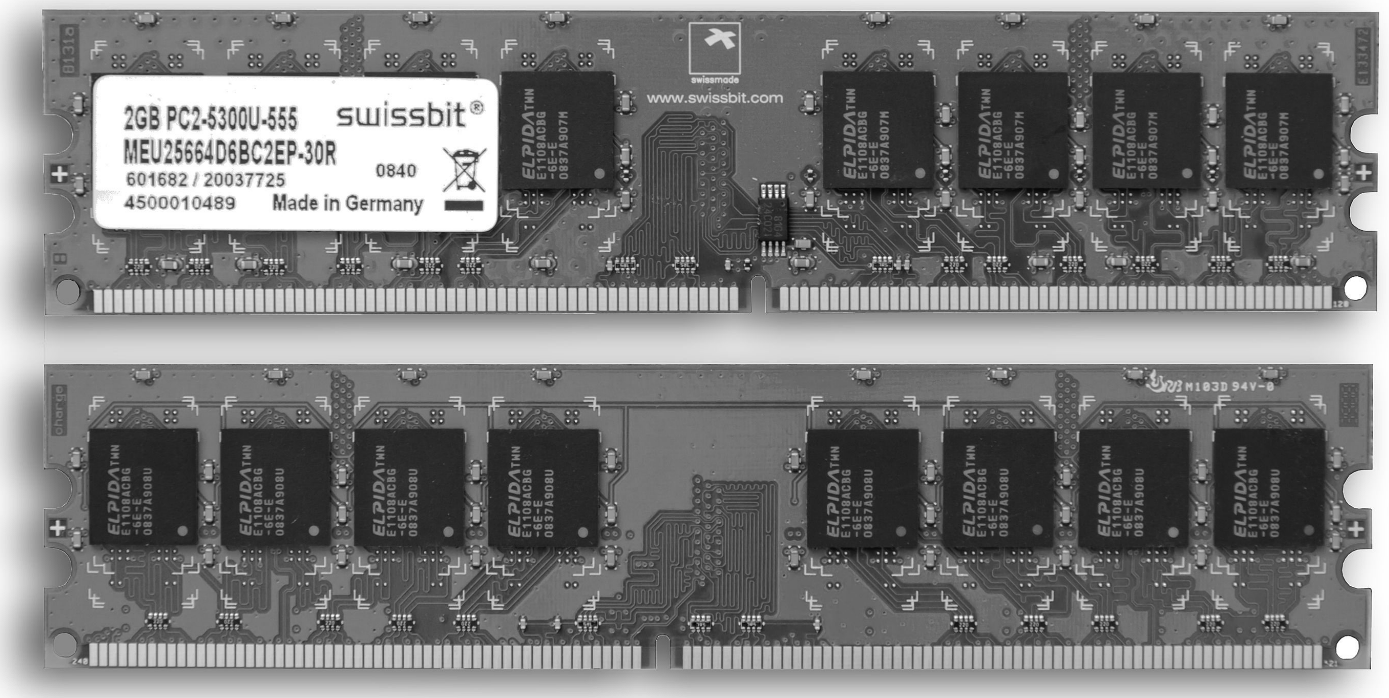
→ many virtual disks (files)



Files for zoom

Files for keynote

Virtualize



Recap:

OS ≈ virtual CPU + virtual memory + virtual disk

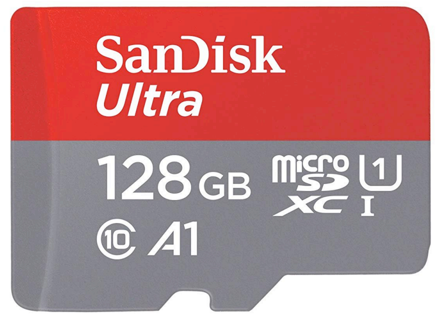
All are one-to-many virtualization here.

Agenda

- Disk
  - SD card driver
  - memory-mapped I/O
- From disk to file system
  - one-to-many virtualization
  - ➡ virtual block store and inodes
  - reading and writing a virtual block store

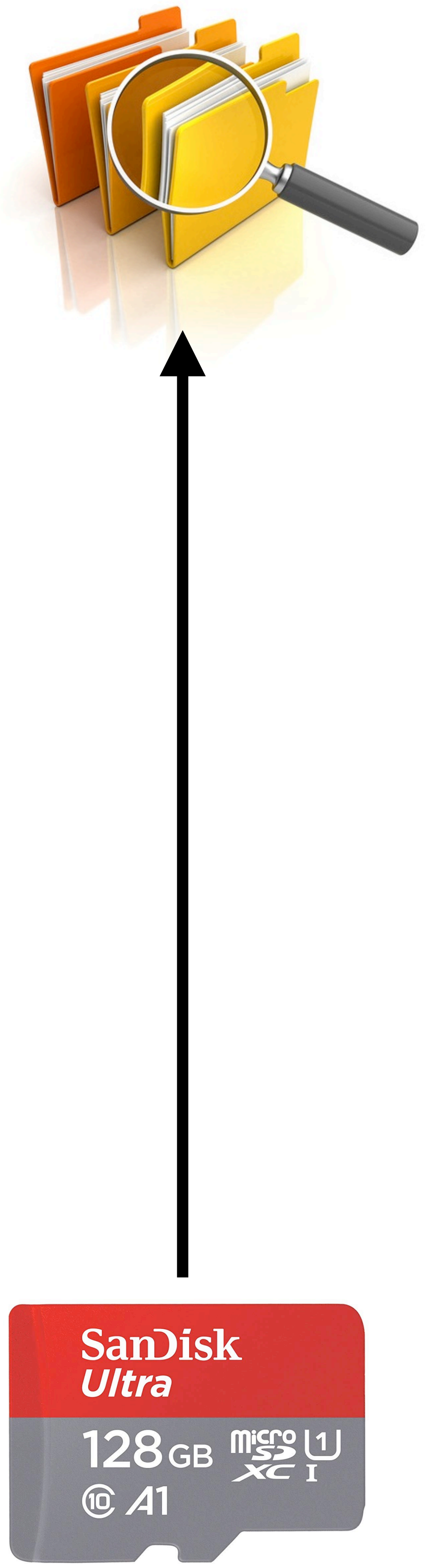
Block store: a sequence of blocks

Content	1st **block**	2nd **block**	3rd **block**	……	2^28th **block**
Address	0	1	2	……	2^28-1



A 128GB disk is a block store with 2^28 blocks

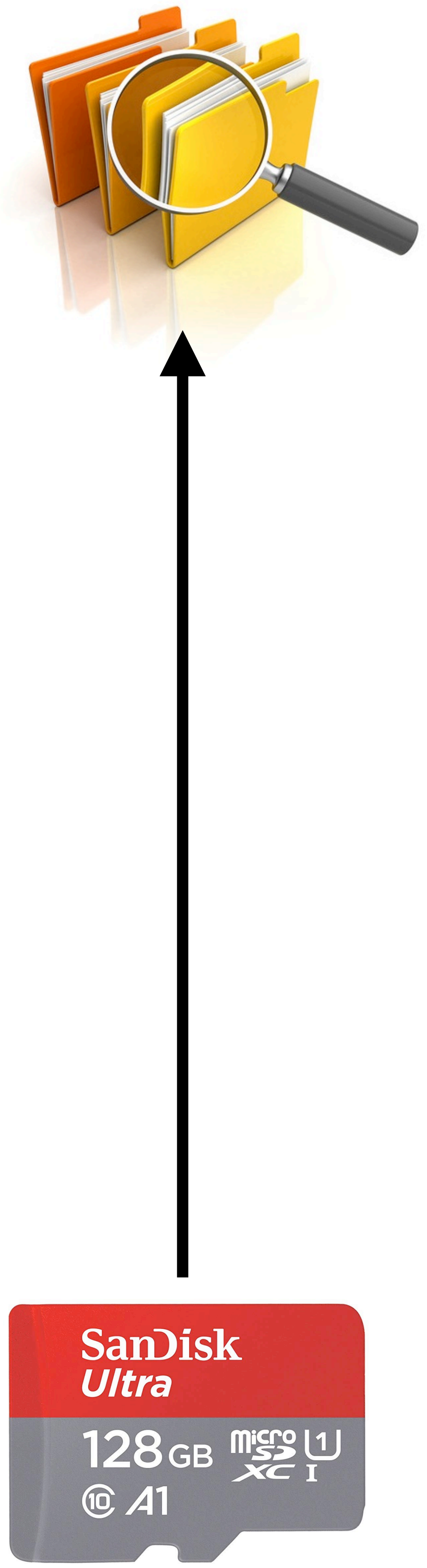
One-to-many virtualization of block store



File system: virtual block stores (VBS)

A 128GB disk is a block store with 2^28 blocks

Example of 256 virtual block stores



File system: virtual block stores (VBS)

VBS #1 (4.2 GB)

Content	1st block	2nd block	3rd block	......	8808038th block
Address	0	1	2	......	8808037

VBS #2 (5 MB)

Content	1st block	2nd block	3rd block	......	10240th block
Address	0	1	2	......	10239

⋮

VBS #256

A 128GB disk is a block store with 2^28 blocks

Virtual block stores as files

File system: virtual block stores (VBS)

VBS #1 (4.2 GB)

Harry Potter movie

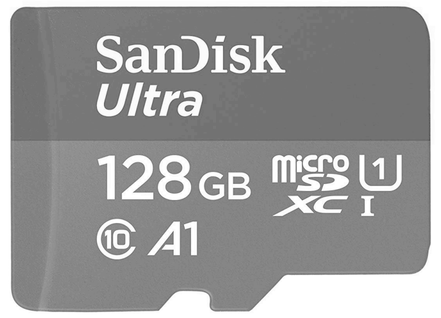
Content	1st block	2nd block	3rd block	......	8808038th block
Address	0	1	2	......	8808037

Something evil has returned to Hogwarts!
Harry Potter
and the Chamber of Secrets

VBS #2 (5 MB)

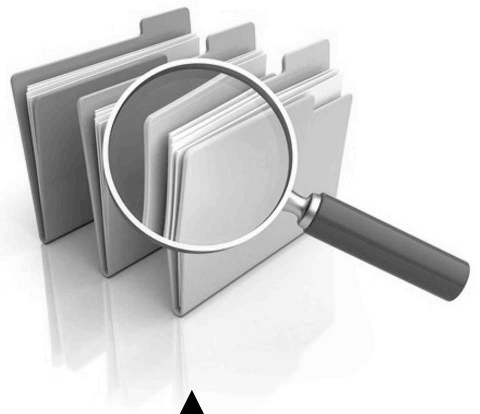
Picture of Yunhao

Content	1st block	2nd block	3rd block	......	10240th block
Address	0	1	2	......	10239



A 128GB disk is a block store with 2^28 blocks

Inode: short term for VBS



File system: virtual block stores (or simply inodes)

Inode #1 (4.2 GB)

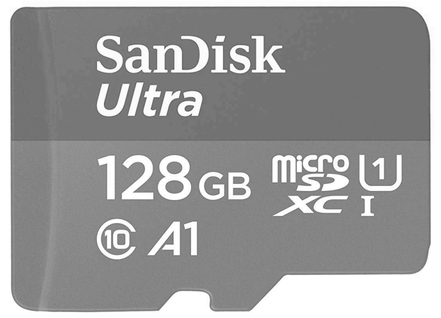
Content	1st block	2nd block	3rd block	......	**8808038th** block
Address	0	1	2	......	8808037

Inode #2 (5 MB)

Content	1st block	2nd block	3rd block	......	**10240th** block
Address	0	1	2	......	10239

⋮

Inode #256



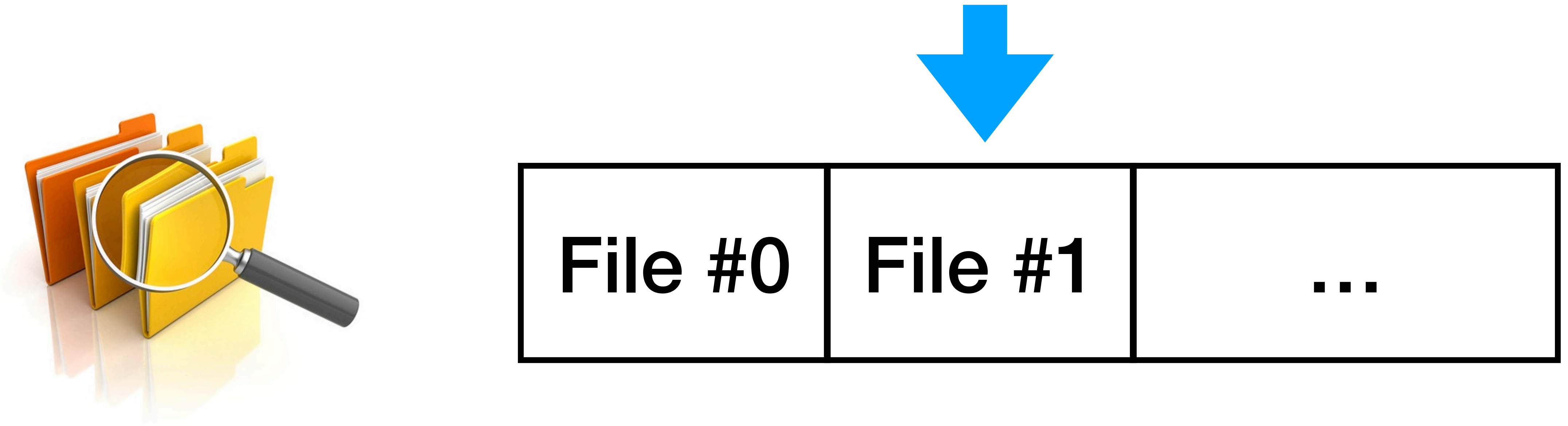
A 128GB disk is a block store with 2^28 blocks

Agenda

- Disk
    - SD card driver
    - memory-mapped I/O
- From disk to file system
    - one-to-many virtualization
    - virtual block store and inodes
    - ➡ reading and writing a virtual block store

Step #1: user reading an inode

1 `Read (ino = 1, offset = 15)`



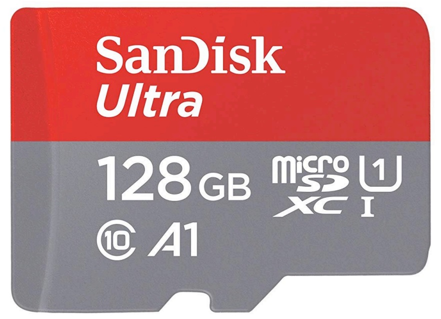
File #0	File #1	...

Step #2: file system reading metadata

1 `Read (ino = 1, offset = 15)`

File #0	File #1	…

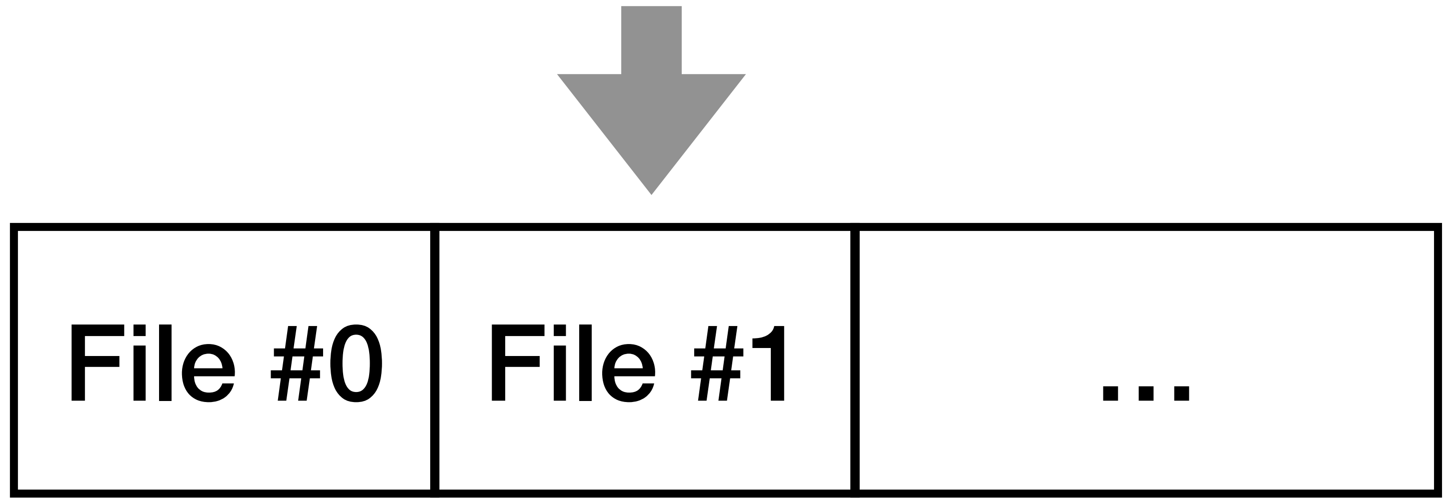
2 `Read the metadata of inode #1`



Super Block	Metadata region	Data region
Block #0	Block #1 … Block #m	Block #m + 1 … Block #n

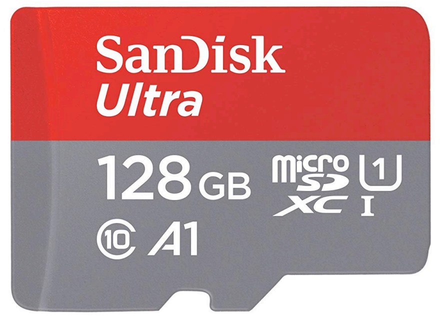
Step #3: file system reading data

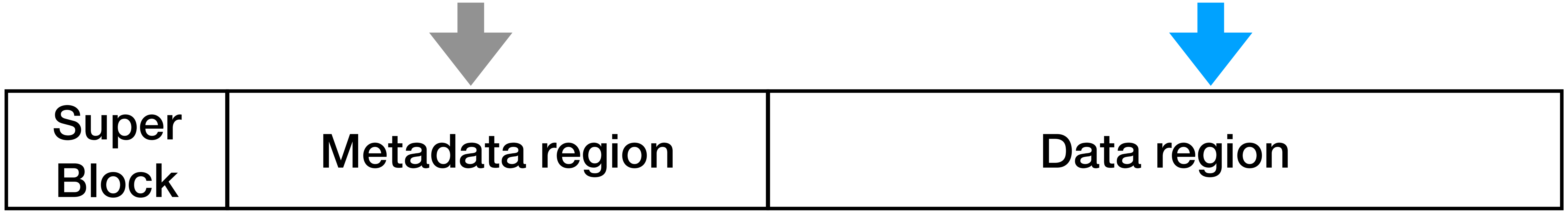
1 `Read (ino = 1, offset = 15)`



File #0 | File #1 | …

2 `Read the metadata of ino` 3 `Read the data of inode #1`





Super Block | Metadata region | Data region

Block #0 Block #1 … Block #m Block #m + 1 … Block #n

Basic file system interface for users

File #0	File #1	...

```
typedef struct inode_store {
    int (*getsize)(struct inode_store *this_bs, unsigned int ino);
    int (*setsize)(struct inode_store *this_bs, unsigned int ino, block_no newsize);
    int (*read)(struct inode_store *this_bs, unsigned int ino, block_no offset, block_t *block);
    int (*write)(struct inode_store *this_bs, unsigned int ino, block_no offset, block_t *block);
    void *state;
} inode_store_t;
```

https://github.com/yhzhang0128/egos-2000/blob/main/library/file/inode.h

P5: A FAT-style file system

- Implement 4 functions:

```
/* below is the SD card block store */
/* ninodes is the “how-many” of one-to-many virtualization */
fatdisk_create(below, below_ino, ninodes);

/* read and write a block of an inode */
fatdisk_read(this_bs, ino, offset, *block);
fatdisk_write(this_bs, ino, offset,*block);

fatdisk_free_file(*snapshot, *fs); /* see next slide */
```

Caution!

- In P5, you implement on-disk data structures
    - 3 steps: read from disk; modify memory; write to disk
    - Many bugs are caused by forgetting this 3-step approach
- How to start P5?

⭐ Read helper function `fatdisk_get_snapshot()`

- which, given an inode number, reads 2 blocks to memory

Homework

- P4 is optional
- P5 is due on Dec. 5
- No class next week: Happy thanksgiving!
- The last lecture on Dec. 2 will be educational