Filesystems

Based on slides by Matt Welsh, Harvard

What is a file system?

- A file system provides a high-level abstraction of a low-level storage device.
  - HDDs
    - Windows: FAT32, NTFS, etc
    - Linux: ext4, JFFS2, etc
  - CD/DVD-ROMs
  - Flash drives (SSDs or USB-drives)
  - Network file systems (eg: NFS)
  - Distributed file systems (eg: GFS/GoogleFS)
  - …

What does a file system provide?

- General purpose file systems provide hierarchical access to the data
  - Windows: C:\Users\John\
  - Linux: /usr/John/

- Specialized file systems may not provide hierarchal access at all.
  - Eg: record-oriented file systems
    - Can find all files tagged with “foo”
    - No hierarchal relationship between two entries

File system operations

- Basic features:
  - Create an empty file or a directory
  - Delete a file or a directory
  - Read a file's content
  - Append content to a file (eg: first-time writes)
  - (re)Write a file's content

File system operations

- Advanced features:
  - Security (who can read? write?)
  - Accounting and quotas – prevent your classmates from hogging the disks
  - Background file backup
  - Indexing and search capabilities
  - File versioning
  - Encryption
  - Automatic compression of infrequently-used files

File storage

- How does a file system actually store a file?
- How do directories know what files they're storing?
- How do files even know their own name?

i-node based file systems

- Many modern file systems are index-based file systems (i-node).
  - We will focus on the UNIX-specific implementation of i-nodes.
  - Windows (NTFS) uses a similar structure.

In the beginning…

- To do anything with a storage device, it must be **formatted** to a file system format.

- In an i-node based filing system, this formatting allocates three regions of space:

In the beginning…

- Freshly after a reformat, the file system contains exactly one directory:

/

- Referred to as the “root” directory.

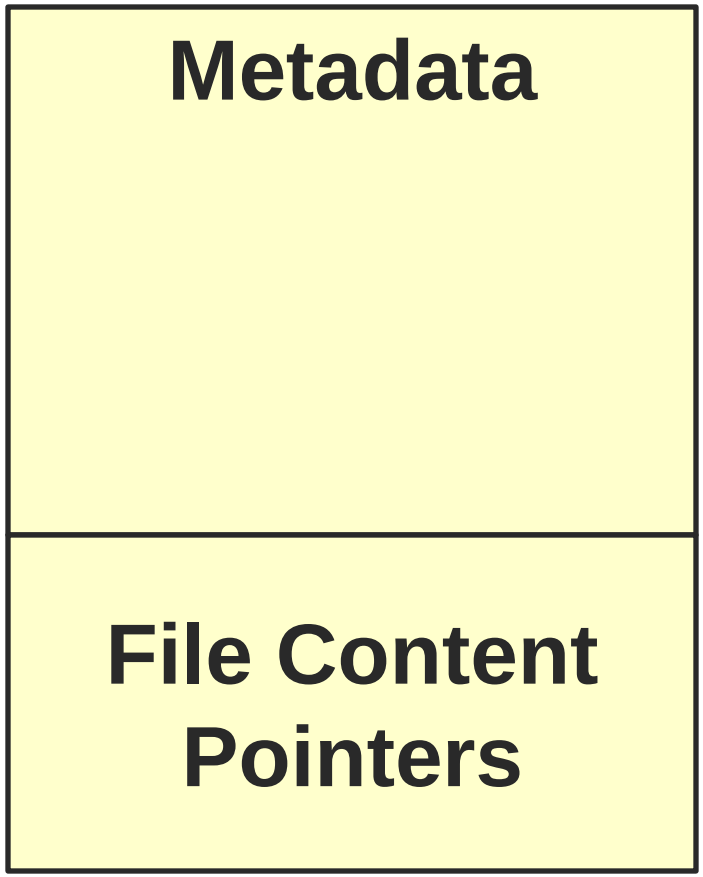
Directory i-node

- The “root” directory is identified by a specifically positioned i-node in the file system (eg: i-node #0).

- An i-node contains information about every object on the disk (files, directories, links, or anything else).

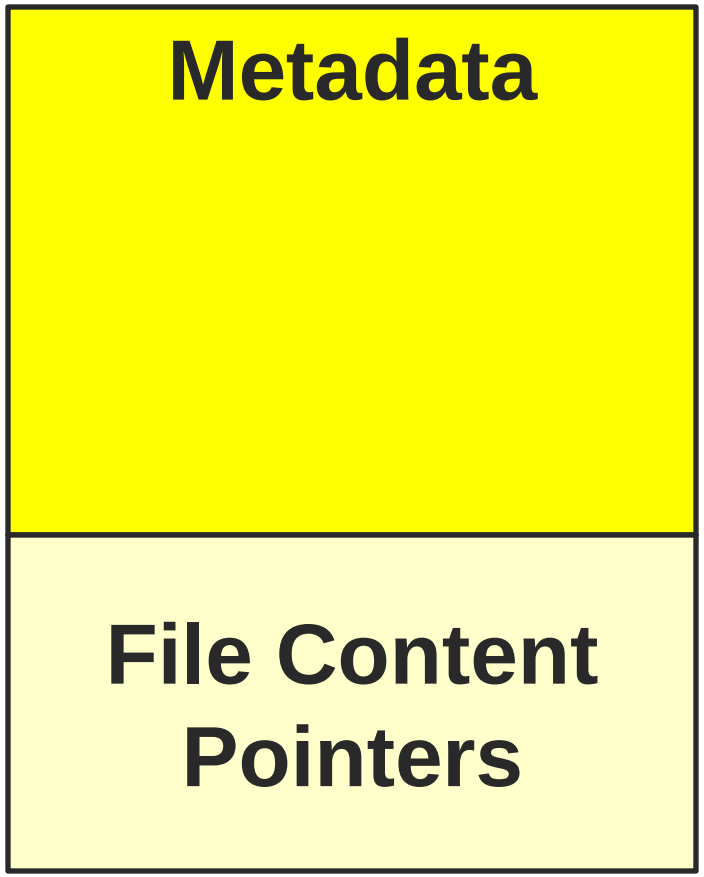
i-node structure

- i-nodes are made up of two main parts:
  - Metadata:
    - Information describing the disk object.
  - File Content Pointers:
    - Disk pointers to the storage of the actual content of the disk object.



i-node structure

- Every i-node contains the same metadata:
  - i-node Number
  - Size
  - Object type
    - Directory? File? Link?
  - Timestamps
    - Creation / Modification / Access Times
  - Security information
  - Link count
  - **NOT** its name!



i-node structure

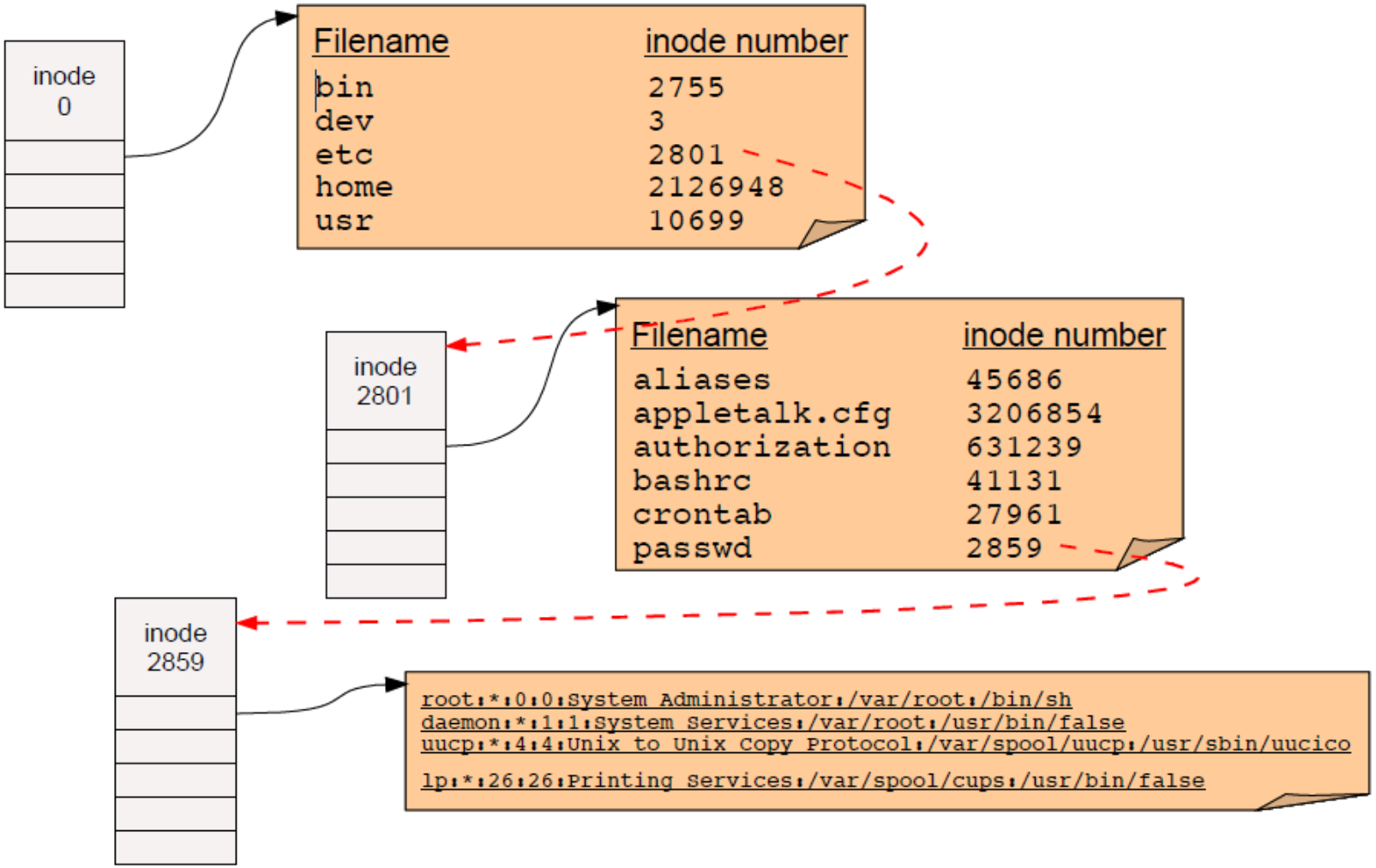
- If the i-node type is a directory, the content section of the i-node describes a “directory file”.

- A “directory file” is simply a list:
  - The name of all the objects contained in the current directory (subdirectories, files, etc).
  - The i-node number of each of the objects.
  - Eg:

dir1	**13**
file1	**34**

Pathname resolution

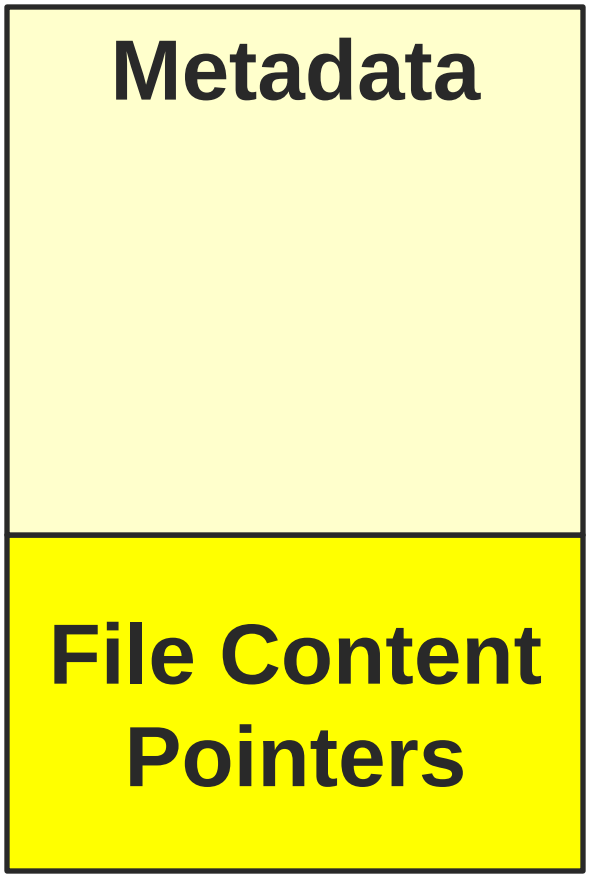
- To look up a pathname "/etc/passwd", start at root directory and walk down chain of inodes...



i-node structure

- The second part of every i-node consists of how we access our data.

- Four types of pointers:
  - Direct
  - Single-indirect
  - Double-indirect
  - Triple-indirect



Direct i-node entries

- Direct pointers point directly to a block on disk and are always used before any indirect pointers are used.
  - If the size of each disk block was 4 KB and there was 10 direct pointers, the first (10 * 4KB) of data would be stored via direct pointers.

- +Efficient access
- -Not very scalable

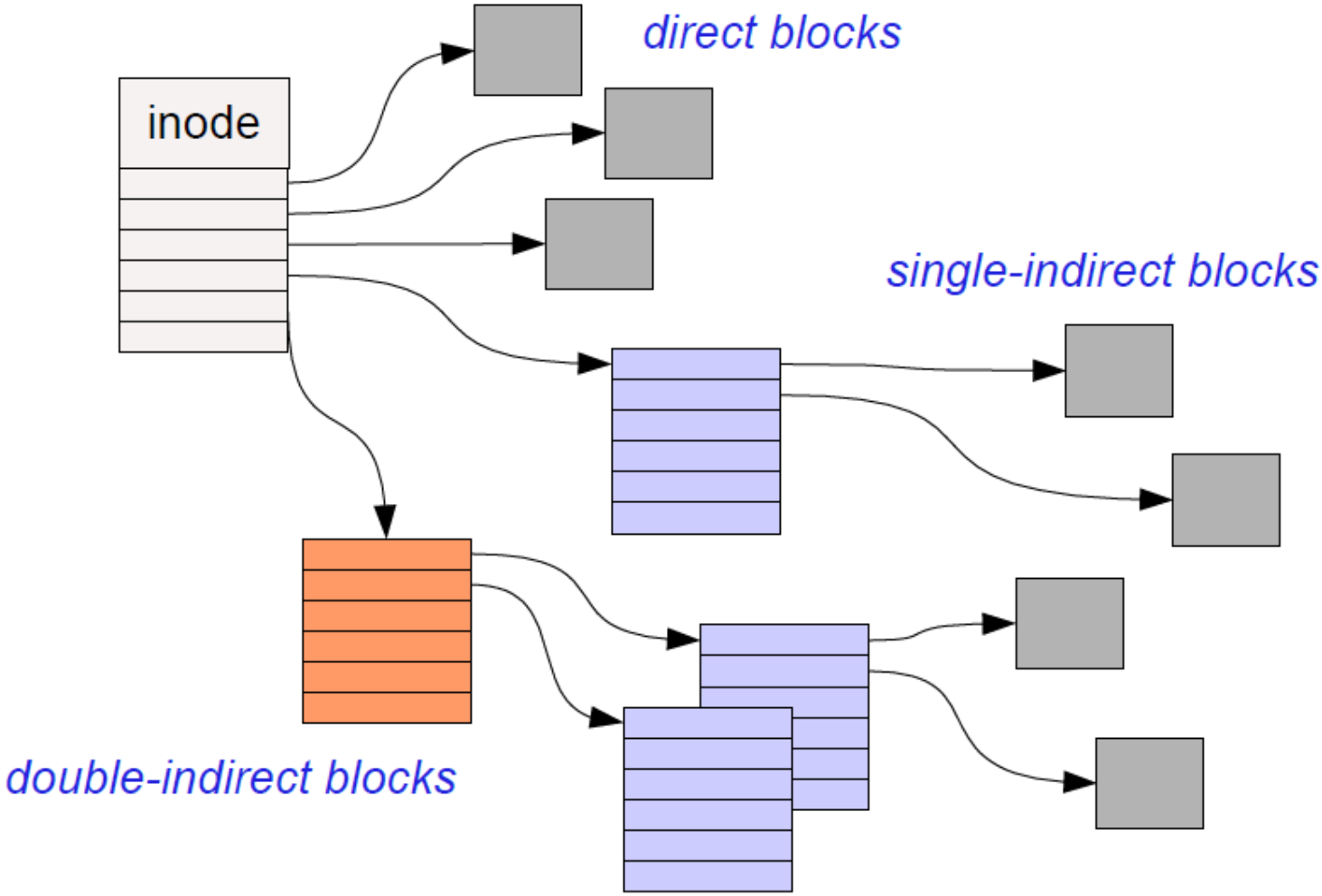
Single-indirect i-node entries

- Instead of pointing directly to the data, single-indirect point to a disk block that is filled with direct pointers.
  - Disk blocks: 4 KB
  - Disk pointer size: 4 B
  - How much could be stored via one single-indirect pointer in an i-node?

Double-indirect i-node entries

- Following the same pattern, double-indirect entries point to a disk block full of single-indirect pointers.
  - How much could be stored via one double-indirect pointer in an i-node?

i-node pointers overview



Stupid directory tricks

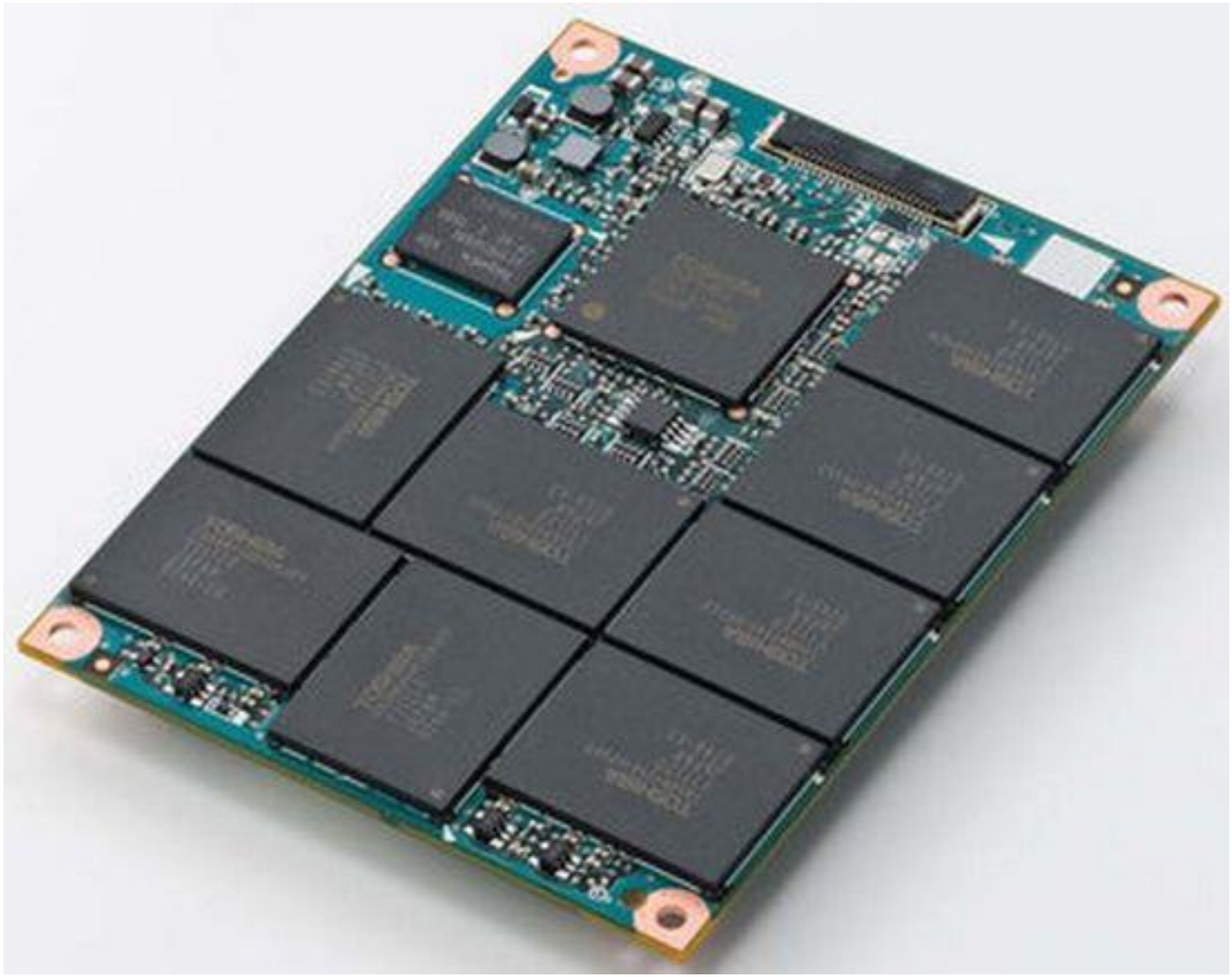
- Directories map filenames to inode numbers. What does this imply?
- We can create multiple pointers to the same inode in different directories
  - Or even the same directory with different filenames
- In UNIX this is called a "hard link" and can be done using "ln"

```
bash$ ls -i /home/foo
287663 /home/foo          (This is the inode number of "foo")
bash$ ln /home/foo /tmp/foo
bash$ ls -i /home/foo /tmp/foo
287663 /home/foo
287663 /tmp/foo
```

  - "/home/foo" and "/tmp/foo" now refer to the same file on disk
    - Not a copy! You will always see identical data no matter which filename you use to read or write the file.
  - Note: This is not the same as a "symbolic link", which only links one filename to another.

Disks

Based on slides by Matt Welsh, Harvard



Physical disks

- File systems are an abstraction above a physical disk device.
  - HDDs (eg: magnetic platters)
  - SSDs (eg: flash/NAND memory)
  - SANs (“Storage Area Networks”)
  - …