CPSC 313: Computer Hardware and Operating Systems

Unit 4: File Systems

From the API to the Disk
(Today is mostly *asking questions*!)

Unit 4: File sytems

- Unit Map:
  - P18: File Systems APIs and How disks work
  - 4.1. Using File Systems APIs
  - P19: File descriptors
  - 4.2. File descriptors management
  - P20: File Systems implementation overview
  - 4.3. How we represent files

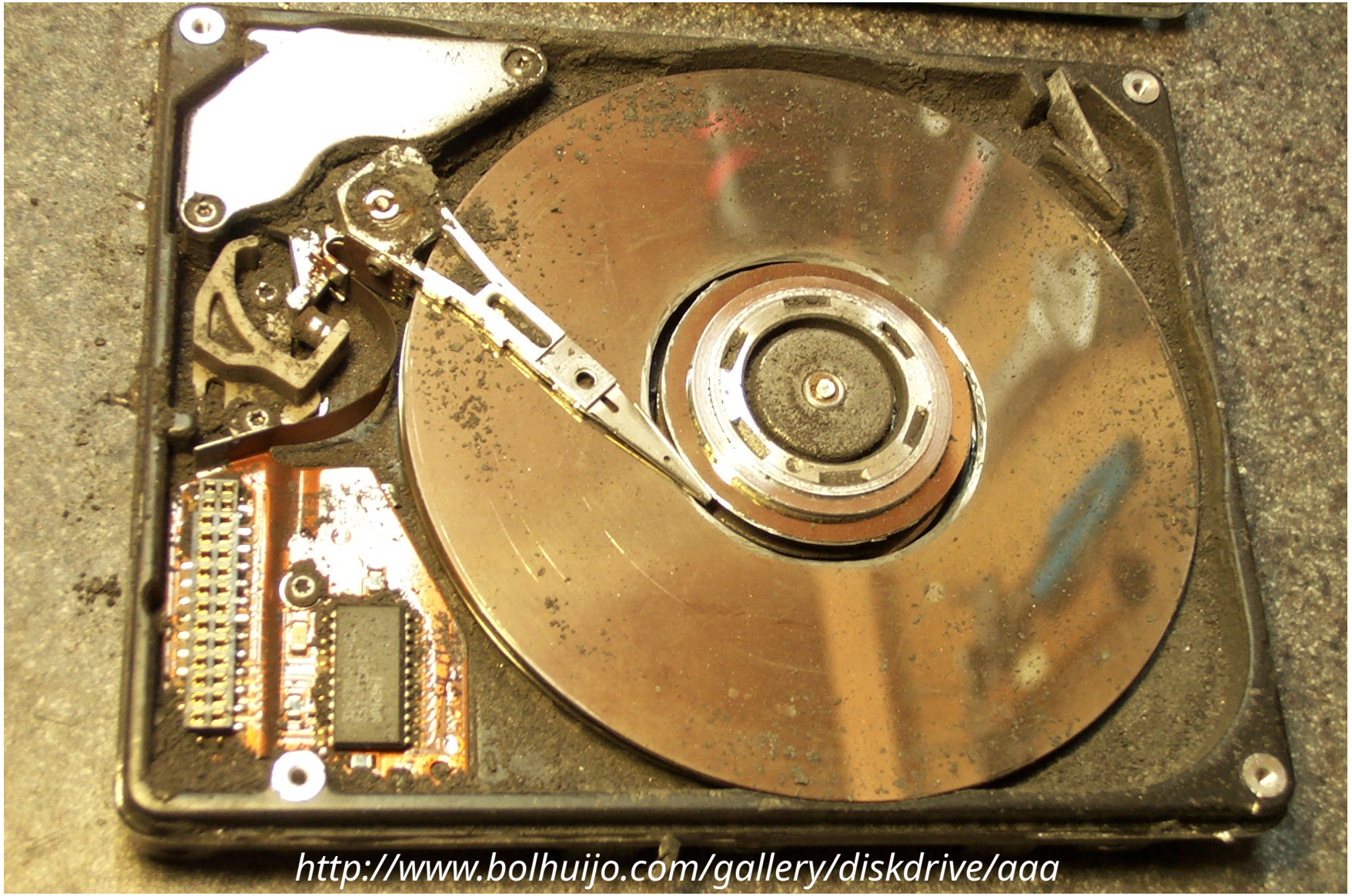
Unit 4: File sytems

- Unit Map (continued):
  - P21: Why fixed-size block file systems?
  - 4.4. Building a file index
  - P22: Getting File System metadata
  - 4.5. Naming
  - P23: Case study: the V6 File System
  - 4.6. File Systems case studies (V6 vs ext2).

Today

- Learning Outcomes:
  - Develop intuition about what has to be done to go from the API to the storage device
  - What data needs to be kept, where does it need to reside, is it persistent or not?

Head Crash



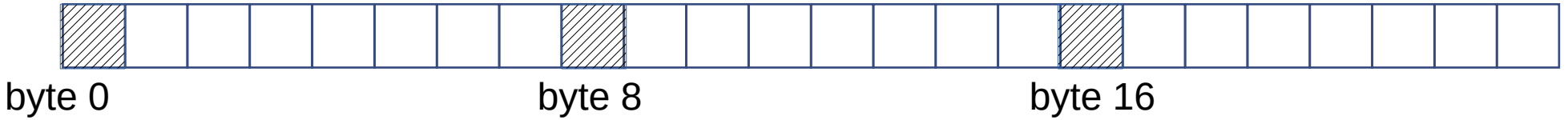
http://www.bolhuijo.com/gallery/diskdrive/aaa

Where we are going

System calls we use to access files: open/close/read/write

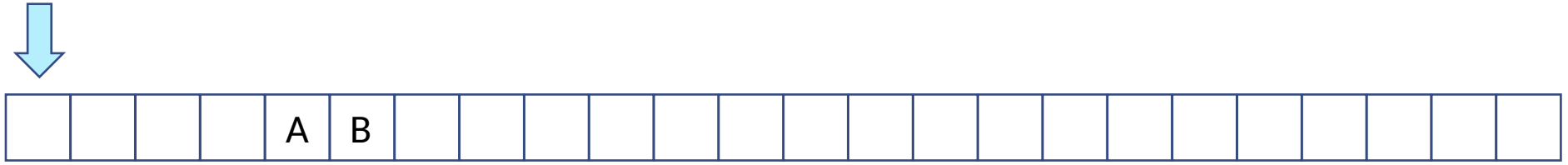
Storage Devices (Disk, SSD)

File – a stream of bytes



- The operating systems maintains an *offset* for each opened file.
- This offset indicates the next location from which to read or to which to write.
- Read and write share the same offset

File – a stream of bytes



- Reading and writing proceeds from the current offset
  - If we read 4 bytes then the offset moves by 4
  - If we write 2 bytes the offset moves by 2

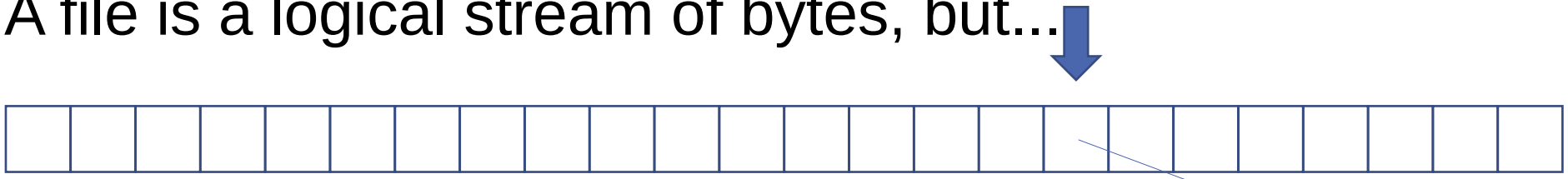
- *What does this tell us about what information the kernel needs to keep and what role the file descriptor plays?*

What we have discovered so far

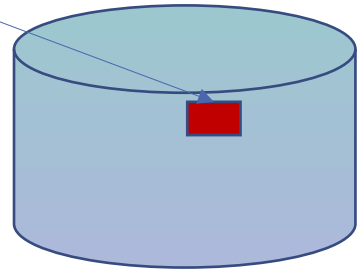
- Each open file needs an offset for reading and writing
- The file descriptor must allow us to find:
  - the file's data
  - the file's metadata (e.g., how large the file is, who can access it, etc)
  - the file offset
- *Are these persistent?*

A stream of bytes → blocks

- A file is a logical stream of bytes, but...



- Data can only be read from disk one block at a time



- Reading a byte requires that we retrieve a block from the disk
  - How do we find the block?

This should seem familiar as the same strategy is used for cache lines

Offset → byte – Which block?

Logical block number (LBN) -- not disk address!

- Offset divided by block size = block number, but …
  - This block number is relative to the start of this file
  - There are lots of files on the disk and lots of blocks/sectors as well

Offset→ byte – Which block?

- What does this tell us about how things are organized?
  - While a file is just a byte stream, we are going to move a file's data to/from disk in units of blocks.
  - So, the file system views the file as a sequence of blocks.
  - We number those blocks starting at 0 and call those logical block numbers (LBN).
  - From the logical block number we have to be able to figure out where the block lives on disk.

What we have discovered so far

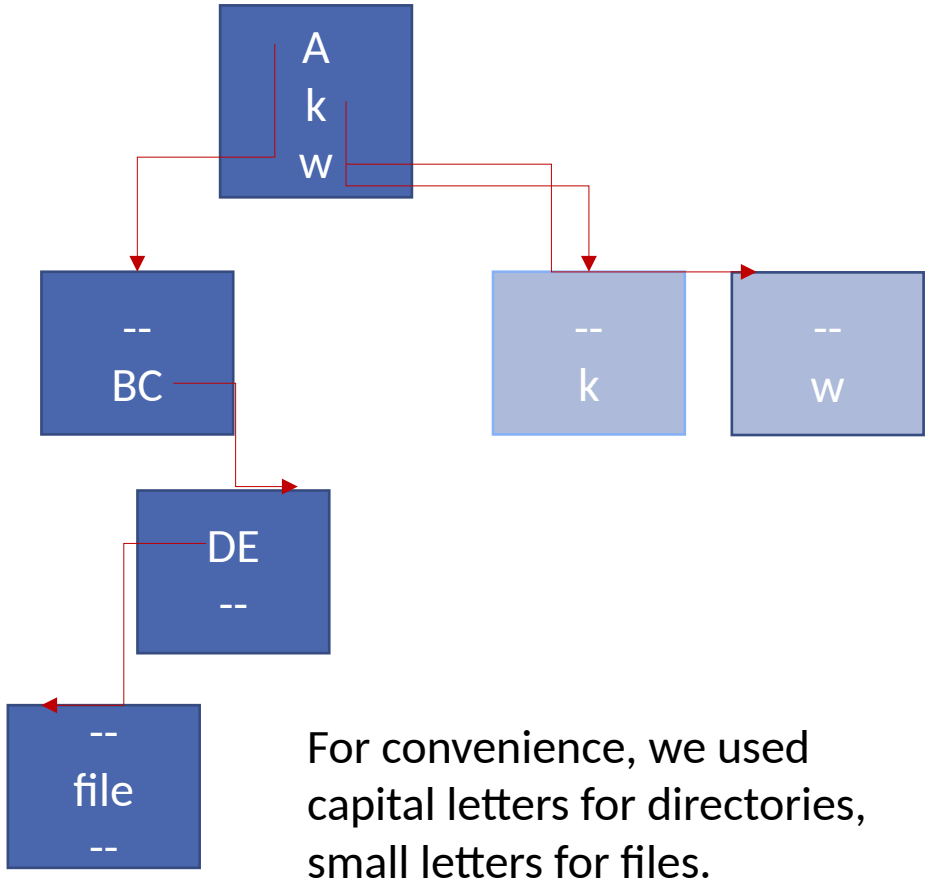
- Each open file needs an offset for reading and writing
- The file descriptor must allow us to find:
  - the file's data;
  - the file's **metadata** (e.g., how large the file is, who can access it, etc);
  - the file offset;
- Must be able to determine to which logical block a byte belongs **arithmetic**
- Must be able to map the logical block to the actual disk location **persistent**

open(filename, flags, mode)

- A file consists of blocks
- We need a map from logical block numbers to disk addresses
- open is given a name. From that name, it must find a data structure that contains the map from LBN to disk address.
- In addition: We need to determine if the process opening the file is allowed to access the file (flags)
  - Need to associate permissions with a file

File name hierarchy

- A file name of the form /A/BC/DE/file
- Consists of directories and files
- Directory maps a name to either another directory or file
- Question –
  - How do we find the first directory?



For convenience, we used capital letters for directories, small letters for files.
(But that's not required!!)

Directory == Folder!

What we have discovered so far

- The file descriptor must allow us to find:
  - the file's data
  - the file's metadata (e.g., how large the file is, who can access it, etc)
  - the file offset
- Must be able to determine the logical block a byte belongs to.
- Must be able to map the logical block to the actual disk location.
- Directories must map names to a file or directory. **persistent**
- Must be able to find the first (root) directory. **it depends …**
- Need to associate access permissions with a file. **persistent**

Some questions about how a file system should behave

- If you and I are both **allowed** to read a file in the file system, should we be able to share the file's data?
- If **two processes** are reading (writing) the same file, should they be using the same file offset?
- If a process **opens** the **same file twice**, should it have one file descriptor or two? In either case, should the two opens share an offset?

More questions about how a file system should behave

- If **two threads** are using the same file descriptor, should they be using the same file offset?
- If one process (the parent) creates another process (a child), should the child **inherit** the parent's file descriptors? Will they use the **same offset**?

Summary

- Some intuition about
  - What information needs to be maintained
  - What information needs to be persistent
- Questions about the behaviour when processes and threads concurrently access the same file