This work is licensed under a Creative Commons license

Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)

You are free to:

Share copy and redistribute the material in any medium or format.

Under the following terms:

Attribution You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial You may not use the material for commercial purposes.

NoDerivatives If you remix, transform, or build upon the material, you may not distribute the modified material.

Volume and File System Analysis

Giovanni Lagorio

giovanni.lagorio@unige.it
https://csec.it/people/giovanni_lagorio
X/GitHub/...: zxgio

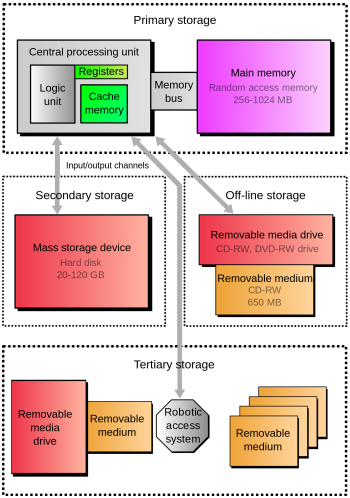
DIBRIS - Dipartimento di Informatica, Bioingegneria, Robotica e Ingegneria dei Sistemi
University of Genova, Italy

Outline

Computer data storage

Hierarchy of storage, we'll deal with secondary/external-memory storage

- not directly accessible by CPU
  - need to transfer desired data to/from main memory
- data transferred in blocks
- significantly slower
- non-volatile



https://commons.wikimedia.org/wiki/File:Computer_storage_types.svg

Storage device terminology

- With removable media, like floppies/CDROMs/DVDs:

  Drive the reader(-writer) device

  Disk the media themselves

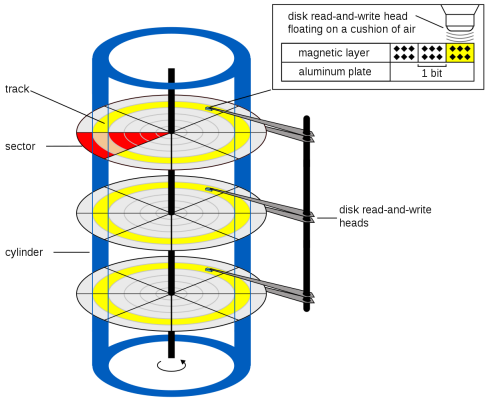
  with fixed-disks, these terms are used interchangeably
- The term disk comes from a time where it made sense; today, we sometimes call "disks" also SD memory cards or SSDs

https://commons.wikimedia.org/wiki/File:Floppy_disk_drive_5_25.jpg

https://commons.wikimedia.org/wiki/File:IBM_DJNA-351520_Hard_Disk_A.jpg

https://commons.wikimedia.org/wiki/File:HardDiskAnatomy.jpg

Disk geometry



- Tracks and sectors are logical, created by low-level formatting
- Sectors could be of any size, typically 512 bytes
  - Modern *Advanced Format Drives (AFD)* have larger sectors
  - Advanced Format 512e, or 512 emulation drives offer legacy interfaces to 512 byte blocks
  - other sizes are possible (but it's unlikely to encounter them today); for the curious, see `https://www.youtube.com/playlist?list=PL18CvD80w43YfBnom7sOSFswQ8lO6zLWu`

`https://commons.wikimedia.org/wiki/File:Hard_drive_geometry_-_English_-_2019-05-30.svg`

Block devices

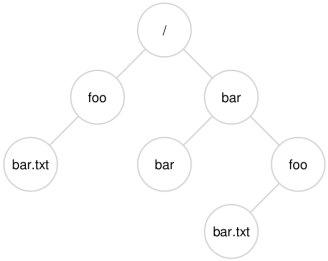
Floppies/HDs/CDs/DVDs/BDs/SD-cards/SSDs/pendrives... are all block devices; following [Car05]'s terminology

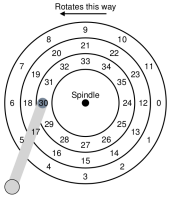
- a volume is a collection of *addressable* blocks
  - these blocks do not need to be contiguous on a physical device
  - a volume can be assembled by merging smaller volumes
- a partition is a *contiguous* part of a volume
  - partitioning is optional: some removable storage does not use it
- by definition, both disks and partitions are volumes

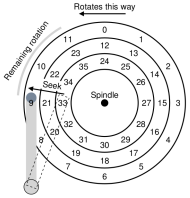
In this part of the course we deal with block-device (forensics) images, like the one acquired from actual devices

File systems

- users do not (directly) deal with blocks, but with files and directories





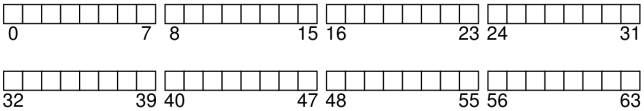


- a file system
  - creates this illusion; i.e., it handles the mapping between files/directories and a collection of blocks (usually, clusters of sectors)
  - consists of on-disk data structures to organize both data and metadata [ADAD18]
- there exist various file-system formats (e.g., FAT, NTFS, . . .)
- (high-level) formatting a volume means to initialize those structures

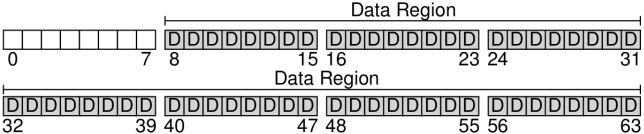
Outline

A toy example: VSFS

VSFS (Very Simple File-System): data and meta-data [ADAD18]



how many *"blocks"* for data vs meta-data?



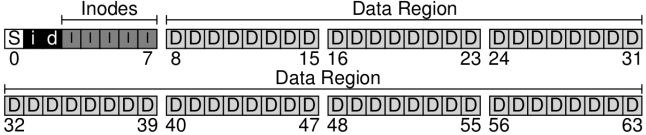
See, for instance, the man page for `mkfs.ext4`

Meta-data

In Unix-like FSs:

- for each "file" there is an inode for storing its metadata (except for names)
  - regular file
  - directory
  - symbolic link
  - FIFO
  - socket
  - character device
  - block device
- how can we keep track of used/free inodes/data-blocks?

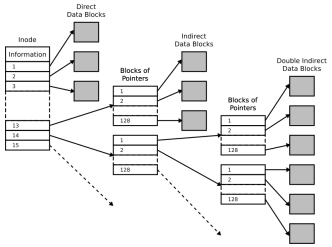
Formatting means preparing: the superblock, i-node/data bitmaps, i-node table, data region:



Inodes

An inode contains

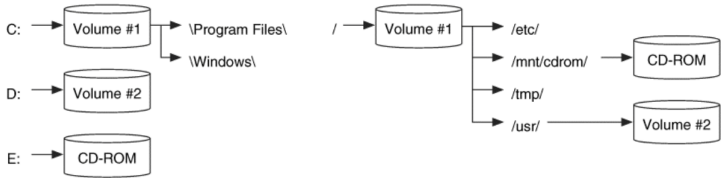
- file type
- UID/GID/permission-bits
- time information
- size in bytes
- number of hard links (AKA names)
- pointers to data blocks



https://commons.wikimedia.org/wiki/File:Ext2-inode.svg

File system mounting

To use the end-user view, a file system, stored on a block device, must be mounted (or "parsed")



Most OSes automatically mount attached devices

Block devices

Block devices can be seen as "files" themselves

Linux special files, typically under `/dev`
(various "aliases" in `/dev/disk`)
`lsblk` lists information about available block devices

Windows prefixes can alter how paths are interpreted

- `\\` introduces the Universal Naming Convention (UNC)
  - usually used for network resources; also for WSL "shares"; e.g., `\\wsl.localhost\Ubuntu-22.04\home\...`
- `\\.\` prefix accesses device namespace instead of file namespace. This is how access to physical disks and volumes is accomplished (with compatible APIs)
  - e.g., in PS, `GET-CimInstance -query "SELECT * from Win32_DiskDrive"`, or `Get-Disk`, `Get-Partition`, ...
- `\\?\` prefix disables parsing and sends the string that follows it straight to the file system
  - E.g., volumes can be accessed via their GUIDs: `start \\?\Volume{`*`Volume-GUID`*`}\`

`https://learn.microsoft.com/en-gb/windows/win32/fileio/naming-a-file`

Mounting image files

Viceversa, (image) files can be seen as block devices...

Linux `losetup` sets up and controls loop devices

- `--list`
- `--find [--show] [--partscan]` *image*
- `--detach[-all]`

then, we can `mount` them.

- `--offset` *off* move the data start; *off* can be followed by `KiB` (=1024), `KB` (=1000), ...

Loop devices can also be automatically created by `mount`:

`mount [-o loop]` *image*

Other important mount options

- `ro`
- `offset=...`

Windows OSFMount, *Arsenal Image Mounter* (free ver.), *FTK Imager*, ...

- `winget install --id PassmarkSoftware.OSFMount`

Unmounting

To avoid losing data/corrupting a FS, it's important to unmount/remove external drives "safely"

https://www.deathbulge.com/comics/131

Crash consistency

Two major approaches

- file-system checker, *before* mounting
- write-ahead logging AKA journaling

Simple ideas, complex implementations [ADAD18]

Mounting partitions

Example/Exercise

1. Decompress `two-partitions.dd.xz` and verify the SHA 256: `de677468d13028548a0c46df5c9426c4bfe9690e` ...
2. Use `fdisk` to list the partitions
3. Create loop devices for each partition with/w.o. `--partscan`
4. Mount each partition with/w.o. a specific loop device
   - and take a look at `hello.txt` and `wikipedia.txt`

EWF

Linux
- `ewfinfo`
- `ewfverify`
- `ewfmount` *`image mount-point`* "opens" the image and exports a corresponding virtual raw ro-image
  - use `umount` to unmount it
- `ewfexport [-t` *`format`*`]` *`image`*

Windows the previously listed tools should handle EWF

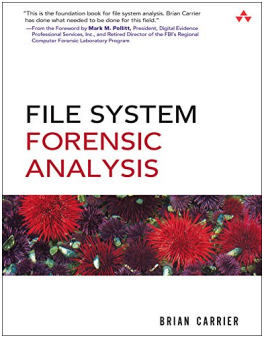
Exercise

1. Verify the integrity of `canon-sd-card.e01` [GFRD09] by using `ewfverify`
2. Calculate the SHA256 of the raw image contained inside `canon-sd-card.e01`

Outline

File System Forensics Analysis

The "bible" for this part of the course is Brian Carrier's "File System Forensics Analsysis" [Car05], where you can find more details



Layers

There are different analysis layers; The Sleuth Kit (TSK) uses the prefixes

- `img_` for images
- `mm` (media-management) for volumes
- `fs` for file-system structures
  - `j` for file-system journals
- `blk` for blocks/data-units
- `i` for inodes, the file metadata
- `f` for file names

Typically followed by

- `stat` for general information
- `ls` for listing the content
- `cat` for dumping/extracting the content

Image layer

- `img_stat` *image* displays details an image
- `img_cat` *image* output contents of an image
  - without options, equivalent to `cat` for raw images

TSK options

We always show only basic options, check the man pages for the full list

(Trivial) Exercise

Check the types of `two-partitions.dd` and `canon-sd-card.e01`

Essential vs Non-essential Data

- At every layer we'll deal with different data structures
  - E.g., for each file, a FS may associate a name to the location of the corresponding data
    - If name or location were incorrect, then the content could not be read
    - OTOH, the last-access time or the data of a deleted file could be correct but we don't know
  - So, name and location are *essential data*
  - While the last-access time is *non-essential data*
- We can trust essential data
- We cannot trust non-essential data without additional evidence

Volume (AKA Media Management) layer

- `mmstat` *image* displays the type of partition scheme
- `mmls` *image* displays the partition layout of a volume
- `mmcat` *image part_num* outputs the contents of a partition

Exercise

For `canon-sd-card.e01`

- Find the type of partition table (`mmstat`)
- List the partitions (`mmls`)
- Extract the `DOS FAT16` partition, by using both
  - `mmcat`
  - `dd` or a dd-like tool
- Check whether the SHA256 of their results match
- Read-only mount the FAT partition and list the files

Outline

Master Boot Record

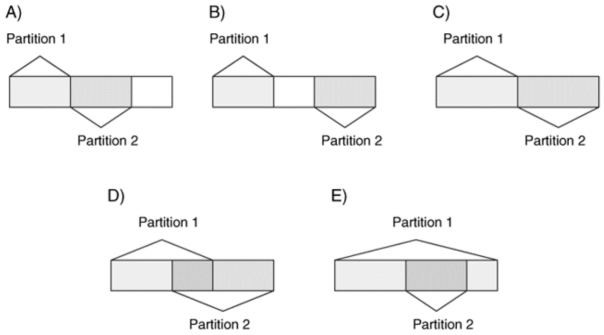
The concept of MBR was introduced in 1983 with PC DOS 2.0.

- boot sector at the very beginning of partitioned media
- it contains
  - machine code for the boot loader, which usually loads and executes the active-partition Volume Boot Record
  - optional disk signature, a 32-bit value that is intended to identify uniquely the disk medium, at 440/0x1b8
  - information on how the disk is partitioned
    - four 16-byte entries, at 446/0x1be
  - the "signature" 0x55 0xaa

There are also some extensions, see `https://en.wikipedia.org/wiki/Master_boot_record`

Partition consistency checks

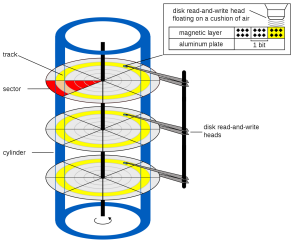
From [Car05]



(A)–(C) are valid, (D) and (E) are typically invalid

CHS and LBA

- Cylinder-Head-Sector is an early method for addressing physical blocks
- CHS uses 3-bytes: 10 bits for cylinders, 8 bits for heads and 6 bits for sectors $\approx$ 8 GB limit
- Replaced by Logical Block Addressing in '90s
  - To convert you need to know the number of heads per cylinder, and sectors per track, as reported by the disk drive
  - Yet, many tools still aligned partitions to cylinder boundaries
- LBA uses 32-bits for starting sectors/total sectors $\approx$ 2 TB limit



Partition entries

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	Master Boot Code															
…																
1A0																
1B0									Disk Signature						Boot ind[1]	Start head
1C0	start Sect[2]	start Cyl[2]	Sys ID[3]	End Head	End sect[2]	End Cyl[2]	Relative Sectors				Total Sectors					
1D0																
1E0																
1F0															55	AA

1. Boot indicator 0x00 = non-boot, 0x80 = bootable
2. Starting sector & starting cylinder are allocated bits, not bytes (0x1C0-0x1C1) same goes for end head and end sector

BIT	0	1	2	3	4	5	6	7	8	9	A		B	C	D	E	F
Value	Starting sector							Starting Cylinder									

3. Common partition values.

0x01	FAT12 <32MB
0x04	FAT16 <32MB
0x05	MS Extended partition using CHS
0x06	FAT16B
0x07	NTFS, HPFS, exFAT
0x0B	FAT32 CHS
0x0C	FAT32 LBA
0x0E	FAT16 LBA
0x0F	MS Extended partition LBA
0x42	Windows Dynamic volume
0x82	Linux swap
0x83	Linux

0x84	**Windows hibernation partition**
0x85	Linux extended
0x8E	Linux LVM
0xA5	FreeBSD slice
0xA6	OpenBSD slice
0xAB	Mac OS X boot
0xAF	HFS, HFS+
0xEE	MS GPT
0xEF	Intel EFI
0xFB	VMware VMFS
0xFC	VMware swap

https://www.writeblocked.org/resources/MBR_GPT_cheatsheet.pdf

Exercise

Exercise: MBR

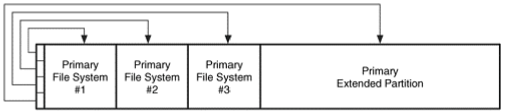
Use ImHex, writing proper patterns, to extract disk and partition information from `mbr{1,2,3}.dd`. Then, answer the following questions:

1. What are the three disk signatures?
2. Is there any MBR with inconsistent partitioning?
3. Are there MBRs without bootable partitions?
4. What is the largest FAT (id=4) partition?
5. Are CHS information always present?

You can ~~cheat~~ check your answers with `fdisk`

Extended partitions

- MBR has only 4 slots for primary partitions
- To work around this limitation, one slot can be used for the primary extended partition, a partition containing other partitions
- Beware of logical-partition addressing, which uses the distance from the beginning of a partition (vs the physical-addressing, from the beginning of the whole disk)



Secondary extended partitions

Inside the primary extended partition we find secondary extended partitions, containing

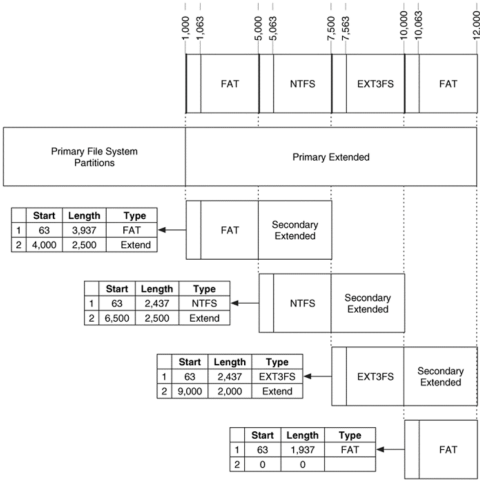
- a partition table t (with the same 512-byte structure) and
- a secondary file-system partition p (AKA logical partition), which contains a FS or other data

t describes where

1. p, w.r.t. t, and the location of the
2. next secondary extended partition (if any), w.r.t. the primary extended partition

For example...

A complete example



Essential data?

Should the following item be considered essential data?

- the boot code
- partition table entries
  - bootable flag
  - partition type
  - CHS addresses
  - LBA address
  - size
- signature (0xAA55)

Exercises

Exercise 1

`ext-partitions.dd` (SHA256: `b075ed83211...`) contains three partition tables: one primary, two extended. Analyze them with ImHex, and compare the result w.r.t. `fdisk/mmls`
Source: `https://dftt.sourceforge.net/test1/index.html`

Exercise 2

Someone purposely damaged the partition table of `hidden-truth.dd` (SHA256: `5f39a8965ec...`)

1. Can you (ro) mount the partitions?
2. Can you repair the broken MBR and mount the deleted partition?
   - hint (ROT13): Lbh pna ernfba nobhg gur ynlbhg be trg fbzr uryc jvgu fvtsvaq (sebz GFX)
3. Can you recover the password protected "secret"?

Outline

UUID/GUID

A Universally/Globally Unique IDentifier (UUID/GUID) is a 128-bit label

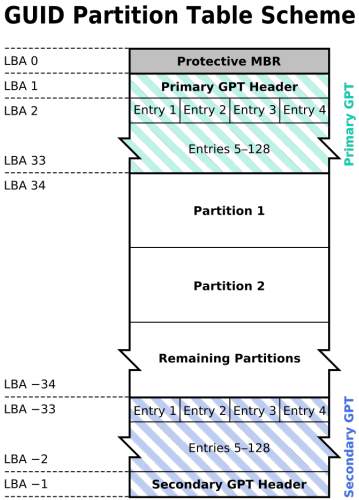
- when properly generated, they are probabilistically unique
- in their canonical textual representation, the 16 octets of a GUID are represented as 32 hex digits, displayed in five groups, in the form 8-4-4-4-12 for a total of 36 characters (32 hex digits and 4 hyphens)
  E.g., `bdeec955-b1b8-44a2-8034-15507d431aca`
- on Linux/WSL you can use `uuidgen` to generate them

See `https://en.wikipedia.org/wiki/Universally_unique_identifier`

Introduction

The GPT format, used by the *Extensible Firmware Interface (EFI)*, which replaced BIOS, is the current standard on PCs; it

- starts with a protective MBR
- supports up to 128 partitions
- uses 64-bit LBA addresses
- keeps "mirrored" backup copies of important data structures



https://commons.wikimedia.org/wiki/File:GUID_Partition_Table_Scheme.svg

GPT header

Byte Range	Description	Essential
0–7	Signature value ("EFI PART")	No
8–11	Version	Yes
12–15	Size of GPT header in bytes	Yes
16–19	CRC32 checksum of GPT header	No
20–23	Reserved	No
24–31	LBA of current GPT header structure	No
32–39	LBA of the other GPT header structure	No
40–47	LBA of start of partition area	Yes
48–55	LBA of end of partition area	No
56–71	Disk GUID	No
72–79	LBA of the start of the partition table	Yes
80–83	Number of entries in partition table	Yes
84–87	Size of each entry in partition table	Yes
88–91	CRC32 checksum of partition table	No
92–End of Sector	Reserved	No

GPT table entry

Byte Range	Description	Essential
0–15	Partition type GUID	No
16–31	Unique partition GUID	No
32–39	Starting LBA of partition	Yes
40–47	Ending LBA of partition	Yes
48–55	Partition attributes	No
56–127	Partition name in Unicode	No

Partition type GUIDs are reserved...

Reserved GUIDs

Operating system	Partition type	Globally unique identifier (GUID)[d] [hide]
—	Unused entry	00000000-0000-0000-0000-000000000000
	MBR partition scheme	024DEE41-33E7-11D3-9D69-0008C781F39F
	EFI System partition	C12A7328-F81F-11D2-BA4B-00A0C93EC93B
	BIOS boot partition[e]	21686148-6449-6E6F-744E-656564454649
	Intel Fast Flash (iFFS) partition (for Intel Rapid Start technology)[41][42]	D3BFE2DE-3DAF-11DF-BA40-E3A556D89593
	Sony boot partition[f]	F4019732-066E-4E12-8273-346C5641494F
	Lenovo boot partition[f]	BFBFAFE7-A34F-448A-9A5B-6213EB736C22
Windows	Microsoft Reserved Partition (MSR)[44]	E3C9E316-0B5C-4DB8-817D-F92DF00215AE
	Basic data partition[44][g]	EBD0A0A2-B9E5-4433-87C0-68B6B72699C7
	Logical Disk Manager (LDM) metadata partition[44]	5808C8AA-7E8F-42E0-85D2-E1E90434CFB3
	Logical Disk Manager data partition[44]	AF9B60A0-1431-4F62-BC68-3311714A69AD
	Windows Recovery Environment[44]	DE94BBA4-06D1-4D40-A16A-BFD50179D6AC
	IBM General Parallel File System (GPFS) partition	37AFFC90-EF7D-4E96-91C3-2D7AE055B174
	Storage Spaces partition[46]	E75CAF8F-F680-4CEE-AFA3-B001E56EFC2D
	Storage Replica partition[47]	558D43C5-A1AC-43C0-AAC8-D1472B2923D1

https://en.wikipedia.org/wiki/GUID_Partition_Table#Partition_type_GUIDs

Exercise

Exercise: GPT

Use ImHex, writing proper patterns, to extract disk and partition information from `gpt.dd`. Then, answer the following questions:

1. What is the disk GUID?
2. How many partitions are there?
3. What are the partition names?
4. Can you find the partition type GUIDs in the previous table?

Outline

Reference model

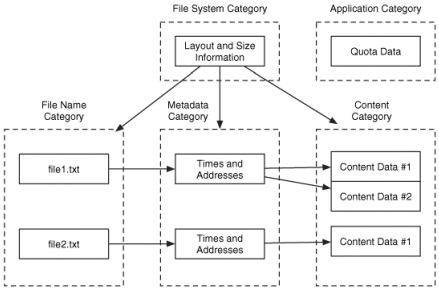
File System FS parameters (size, block size, ...); a "map" to the other structures

Content The actual data, stored in clusters/blocks/data-units

MetaData Data that describes files: size, creation date, ...

File Name Data that assign names to files

Application Data not needed for reading/writing a file; e.g., user quota statistics or a FS journal



TSK is based on these categories

File System Category

To get the general details of a file-system

- `fsstat [-o sect_offs] image`

The output is file-system specific

Stars (*) used to print a hierarchy; e.g., fields listed after ** are inside *

Exercise

1. Find the OEM Name and Volume Label (Boot Sector) in `canon-sd-card.e01`
2. Check whether the partition types are correctly set inside `two-partitions.dd`

String search (first part)

Exercise

1. look for the strings
   - `didattica`
   - `wDeek`
   - `tool`
   - `secret`

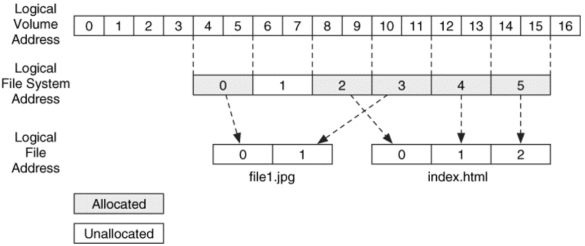
   inside the image file `two-partitions.dd`
2. (ro) mount its partitions, and look for the same strings inside the contained files

Do some string appear only in one of the two searches?
Can you guess why?

Content addressing

Each sector can have multiple addresses, relative to the start of the...

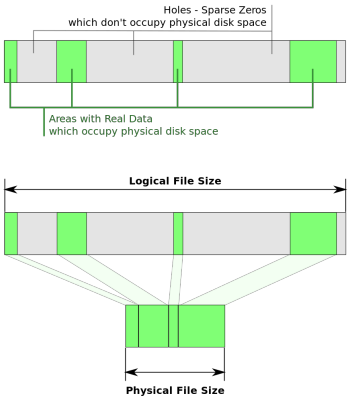
- storage media: physical address
- volume: (logical) volume address
- FS [data area]: (logical) FS address AKA (logical) cluster number
- file: (logical) file address AKA virtual cluster numbers



Files can be fragmented and sparse...

Sparse files

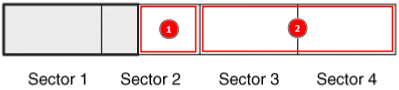
→ `examples/sparse_files`



`https://commons.wikimedia.org/wiki/File:Sparse_file_(en).svg`

Slack space

Example: writing a 612 byte file, in a FS with 2k clusters (and 512-byte sectors)



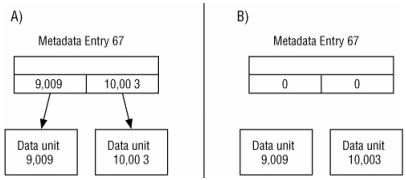
the red parts are slack space; however, an OS

- must write sector (1 and) 2
- might not wipe or even write (2), i.e., sector 3 and 4

Metadata category

During an investigation, it is helpful to search for evidence in deleted files; two major approaches

1. Metadata-based

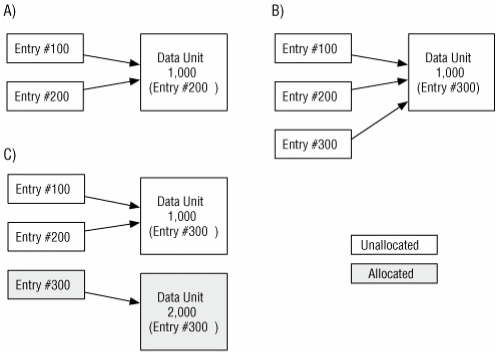


   An orphan is a deleted file for which we cannot determine the full path

2. When metadata is not available: application-based AKA Carving
   - Typically from un-allocated space
   - Does not need any FS information

When recovering files it can be difficult to detect when a data-unit has been reallocated, examples. . .

Where do data-units come from?



- **A** #100 deleted, then #200 created and deleted
  - time information/data-unit content may help to disambiguate
- **B** then #300 created and deleted
- **C** then #300 created and assigned a different data unit

TSK metadata commands (1/2)

- `ils [-o sect_offs] image` — list inode information
  - by default, `-r`, lists only inodes for removed files
  - `-a`, lists only allocated inodes
  - `-m` displays inode details to be fed into `mactime` to produce a timeline; "The good thing about standards is that there are so many to choose from" (A.S. Tanenbaum):
    - Modified Accessed metadata-Changed Birth/creation
    - Modified Accessed Creation Entry-modified
- `istat [-o sect_offs] image inum` — displays details of a meta-data structure (AKA inode)

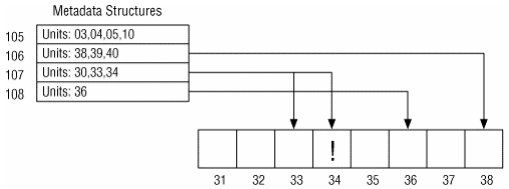
inodes in TSK

TSK always uses the inode abstraction, even when the underlying FS does not natively have them

TSK metadata commands (2/2)

Viceversa, to find the inode corresponding to a data-unit or file name:

- `ifind [-n filename] [-d data-unit] [-o offset] image`



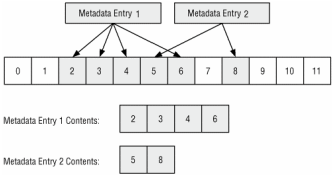
E.g., if `strings -t d ...` gives you some offset n, then
`ifind -d $((n/block-size)) ...` returns the inode number

We'll discuss the name layer later, however (spoiler ☺):

- `ffind [-o sect_offs] image inum` — lists the names using the inode (useful when names are not inside the "inode")

From metadata to contents

- `icat [-o sect_offs] image inum` — displays the content of the file corresponding to the inode number; other arguments:
  - `-s` show slack too
  - `-r` try to recover deleted files

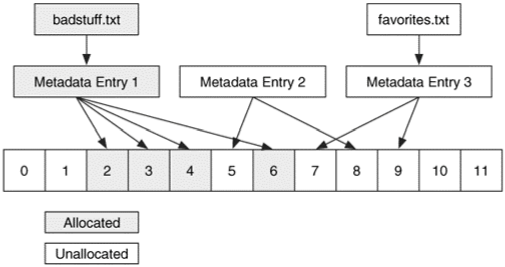


Note: deleted content may be present in unallocated data (without metadata pointing to it).
To check/dump blocks:

- `blkstat image block` shows the details
- `blkcat image block [how-many-blocks]` dumps the block(s)
  (`blkls` lists or outputs blocks too)

Filename Category

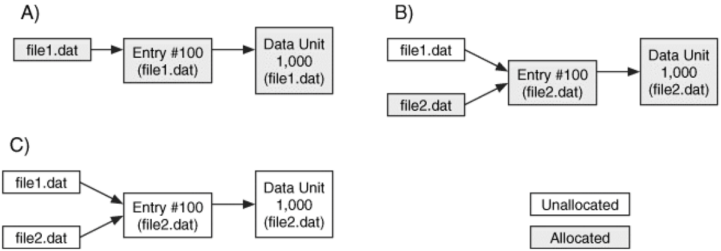
- `fls [-o sect_offs] image [inum]` — list files inside the directory corresponding to the inode number



- `ffind [-o sect_offs] image inum` — lists the names using the inode (useful when names are not inside the "inode")

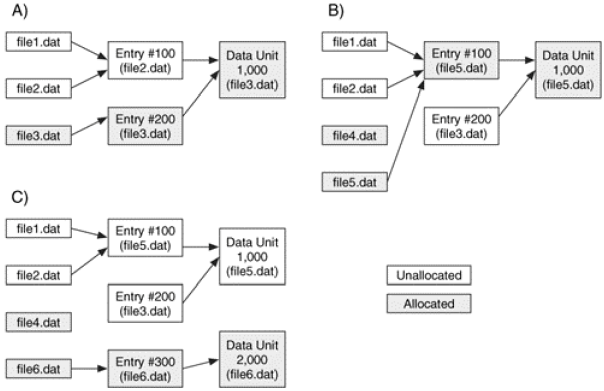
We can't always trust the names, for example...

Out-of-sync filenames (1/2)



- A `file1.dat` created
- B `file1.dat` removed (pointer not wiped), `file2.dat` created
- C `file2.dat` removed

Out-of-sync filenames (2/2)



- A `file3.dat` created
- B `file3.dat` removed, (`file4.dat` created), `file5.dat` created
- C `file5.dat` removed, `file6.dat` created

String search (second part)

Example/Exercise

Let's find out why some of the following strings appear in one search and not the other

- `didattica`
- `wDeek`
- `tool`
- `secret`

Carving

General idea:

- overwritten/damaged/nonexistent metadata
- but known headers and/or footers for some file types
  E.g., `0xFF 0xD8` and `0xFF 0xD9` for JPEG files

Then, you can

1. find all headers
2. find all footers
3. extract in-between sectors

Problems:

- fragmented files
- overwritten data

To apply these techniques: `sigfind` (from TSK), Binwalk, Foremost, ...

Exercises

Exercise

Inside `eighties.dd` (`cc121c3a037f904a4fa5ef51263df9fdb800d89af7330df22615802b81821f9d`) there is a FAT file system with some deleted content.
In particular, there were files with the following SHA256 hashes:

- `4410aaee5ae15917c064f80a073ec75260482b7035fad58c85f1063d0b795733`
- `1b756ad00ad842c3356c093583e2e4fab2540e15ca88750606f45f7efd1f4d26`
- `592f47dfcbeda344fc394987b6e02a65a35d4d849d35d2fc821e5be1889c645d`
- `8a461036c70736eb4ca83e9062318c8293be2baad1c475c41c1945221559048e`
- `0d176b77f6b81468eb7ba367d35bdcbd8fdfc63445c2cc83c5e27c5e0b4c1a14`

Can you recover and identify them?

Note: we'll cover FAT at a later time, you don't need to know anything about FAT internal structure to tackle this exercise

References

[ADAD18] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau.
Operating Systems: Three Easy Pieces.
Arpaci-Dusseau Books, 1.00 edition, August 2018.

[Car05] Brian Carrier.
File System Forensic Analysis.
Addison-Wesley Professional, 2005.

[GFRD09] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt.
Bringing science to digital forensics with standardized forensic corpora.
Digital Investigation, 6:S2–S11, 2009.
The Proceedings of the Ninth Annual DFRWS Conference.