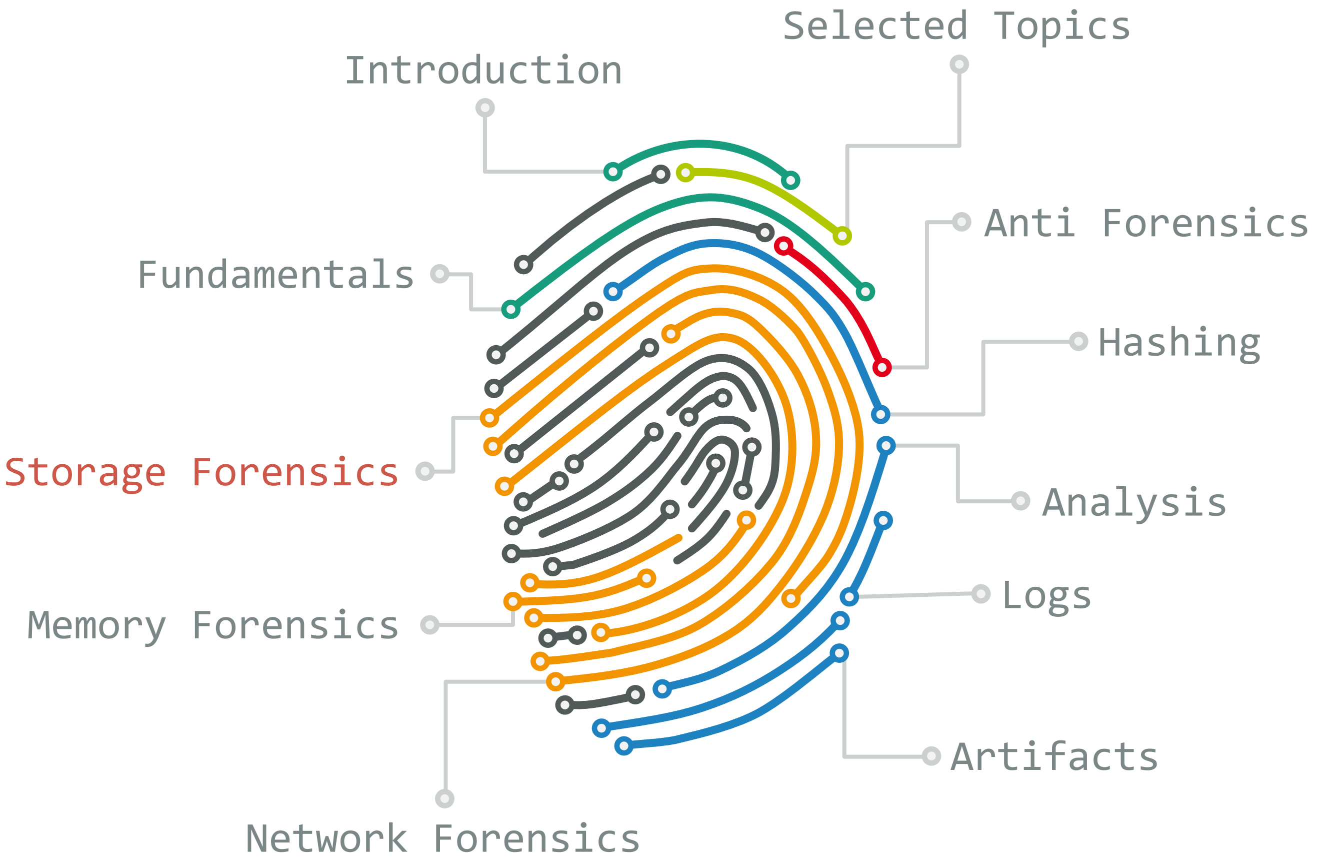
Prof. Dr. Elmar Padilla et al.

Digitale Forensik

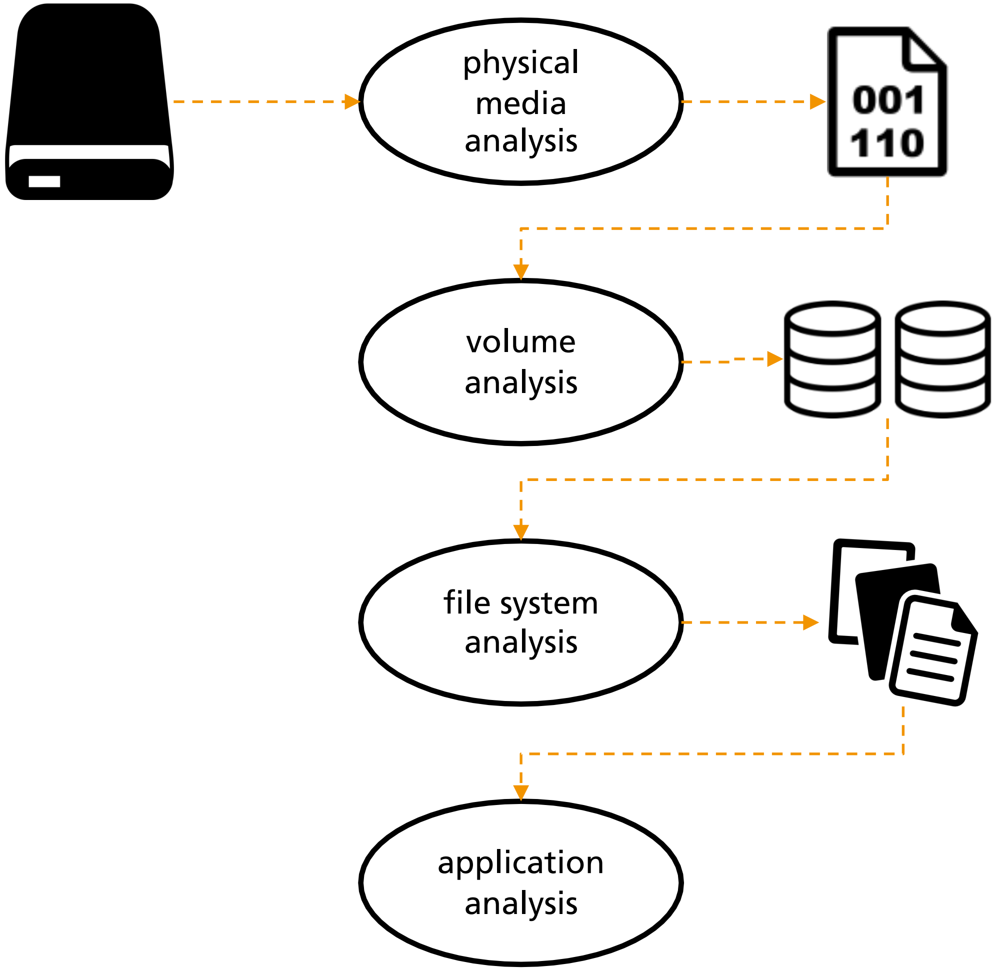
02 - Storage Forensics



File System Forensic Analysis

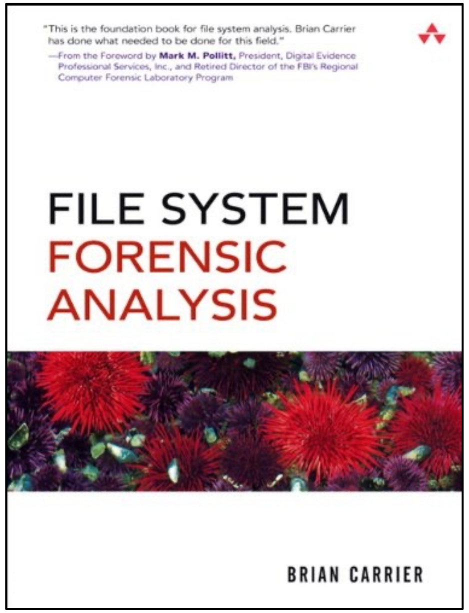
Dr. Brian Carrier

Basis Technology



Sleuth Kit

Autopsy



"This is the foundation book for file system analysis. Brian Carrier has done what needed to be done for this field."

—From the Foreword by **Mark M. Pollitt,** President, Digital Evidence Professional Services, Inc., and Retired Director of the FBI's Regional Computer Forensic Laboratory Program

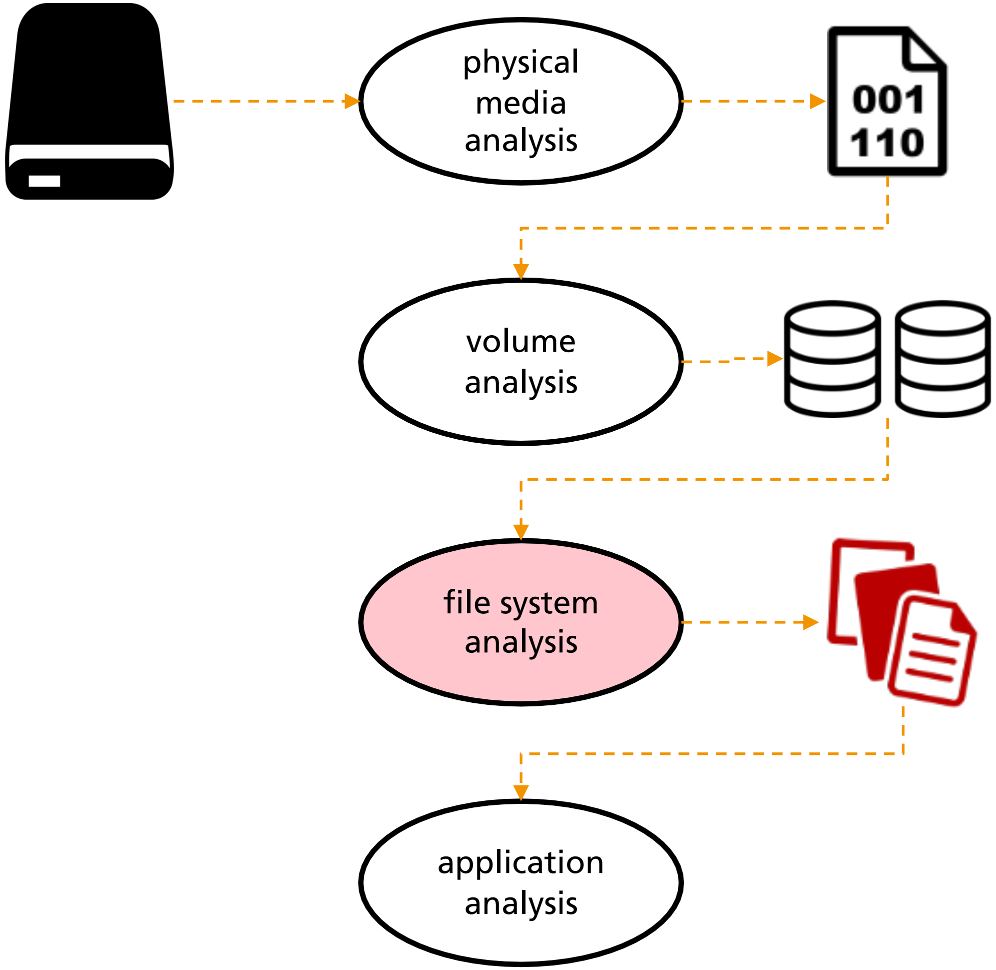
FILE SYSTEM FORENSIC ANALYSIS

BRIAN CARRIER

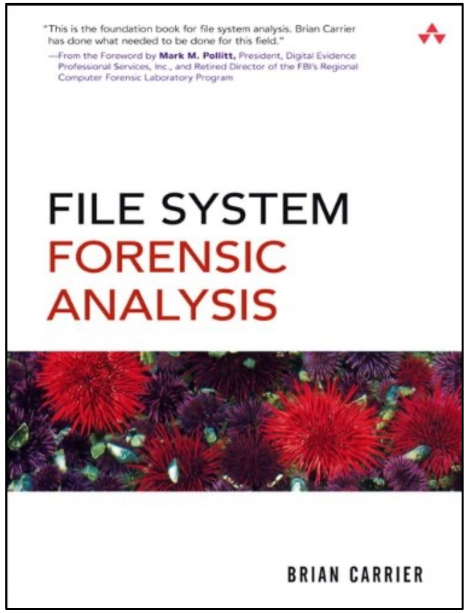
File System Forensic Analysis

Dr. Brian Carrier

Basis Technology



Sleuth Kit



"This is the foundation book for file system analysis. Brian Carrier has done what needed to be done for this field."

—From the Foreword by **Mark M. Pollitt,** President, Digital Evidence Professional Services, Inc., and Retired Director of the FBI's Regional Computer Forensic Laboratory Program

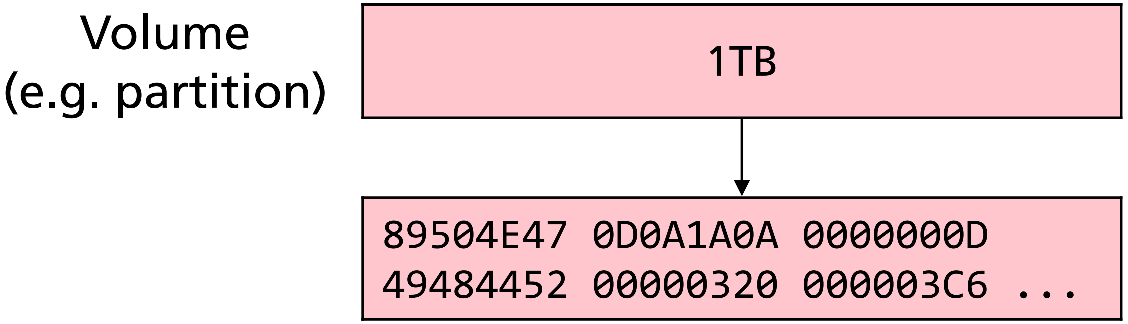
FILE SYSTEM FORENSIC ANALYSIS

BRIAN CARRIER

Autopsy

File System Analysis

Why File Systems?



Volume
(e.g. partition)

1TB

```
89504E47 0D0A1A0A 0000000D
49484452 00000320 000003C6 ...
```

Storing data directly on a volume works, but is infeasible for multiple files

1. How to reference a file?
2. How to quickly find the start of a file and collect metadata (such as timestamps)?
3. How to organize available space?
4. How to implement features such as logging?
5. How to organize the aforementioned concepts?

image.dd

```
89504E47 0D0A1A0A 0000000D 49484452 00000320
000003C6 08060000 001B6DFE F7000000 01735247
4200AECE 1CE90000 00786558 49664D4D 002A0000
00080004 011A0005 00000001 0000003E 011B0005
00000001 00000046 01280003 00000001 00020000
87690004 00000001 0000004E 00000000 00000090
00000001 00000090 00000001 0003A001 00030000
00010001 0000A002 00040000 00010000 0320A003
00040000 00010000 03C60000 0000B58D 47E10000
00097048 59730000 16250000 16250149 5224F000
00400049 696d6167 6530312e 6a7067D4 2455792E
AB5A88C6 2427829A 93758C03 338D3F47 D7D1ACA5
E255FEA7 D144136F 048C0283 0B9DC623 B2B29418
40D1DC83 D224F1DC 08842162 7E04951E 24FCCD44
63927382 F43080C6 242B8A37 1A4D0CF7 7DBABFB7
A7BEEABD DFBDAB6A 5777757F CFCBDADF AE7AFFF7
```

File Systems

„The motivation behind a file system is fairly simple: computers need a method for the long-term storage and retrieval of data. File systems provide a mechanism for users to store data in a hierarchy of files and directories. A file system consists of structural and user data that are organized such that the computer knows where to find them. In most cases, the file system is independent from any specific computer."

Dr. Brian Carrier

image.dd

```
89504E47 0D0A1A0A 0000000D 49484452 00000320
000003C6 08060000 001B6DFE F7000000 01735247
4200AECE 1CE90000 00786558 49664D4D 002A0000
00080004 011A0005 00000001 0000003E 011B0005
00000001 00000046 01280003 00000001 00020000
87690004 00000001 0000004E 00000000 00000090
00000001 00000090 00000001 0003A001 00030000
00010001 0000A002 00040000 00010000 0320A003
00040000 00010000 03C60000 0000B58D 47E10000
00097048 59730000 16250000 16250149 5224F000
00400049 696d6167 6530312e 6a7067D4 2455792E
AB5A88C6 2427829A 93758C03 338D3F47 D7D1ACA5
E255FEA7 D144136F 048C0283 0B9DC623 B2B29418
40D1DC83 D224F1DC 08842162 7E04951E 24FCCD44
63927382 F43080C6 242B8A37 1A4D0CF7 7DBABFB7
A7BEEABD DFBDAB6A 5777757F CFCBDADF AE7AFFF7
```

File Systems

„The motivation behind a file system is fairly simple: computers need a method for the long-term storage and retrieval of data. File systems provide a mechanism for users to store data in a hierarchy of files and directories. A file system consists of structural and user data that are organized such that the computer knows where to find them. In most cases, the file system is independent from any specific computer.“

Dr. Brian Carrier

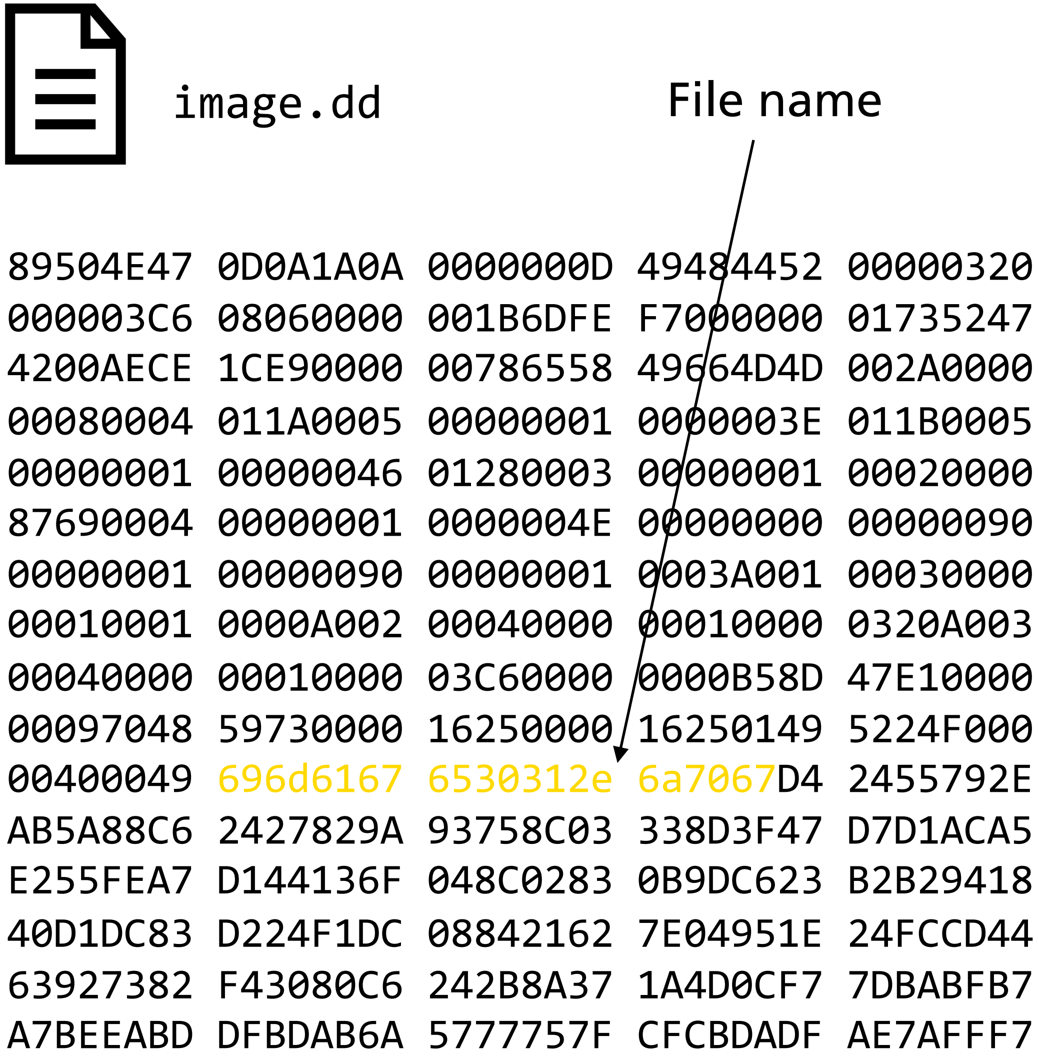


image.dd

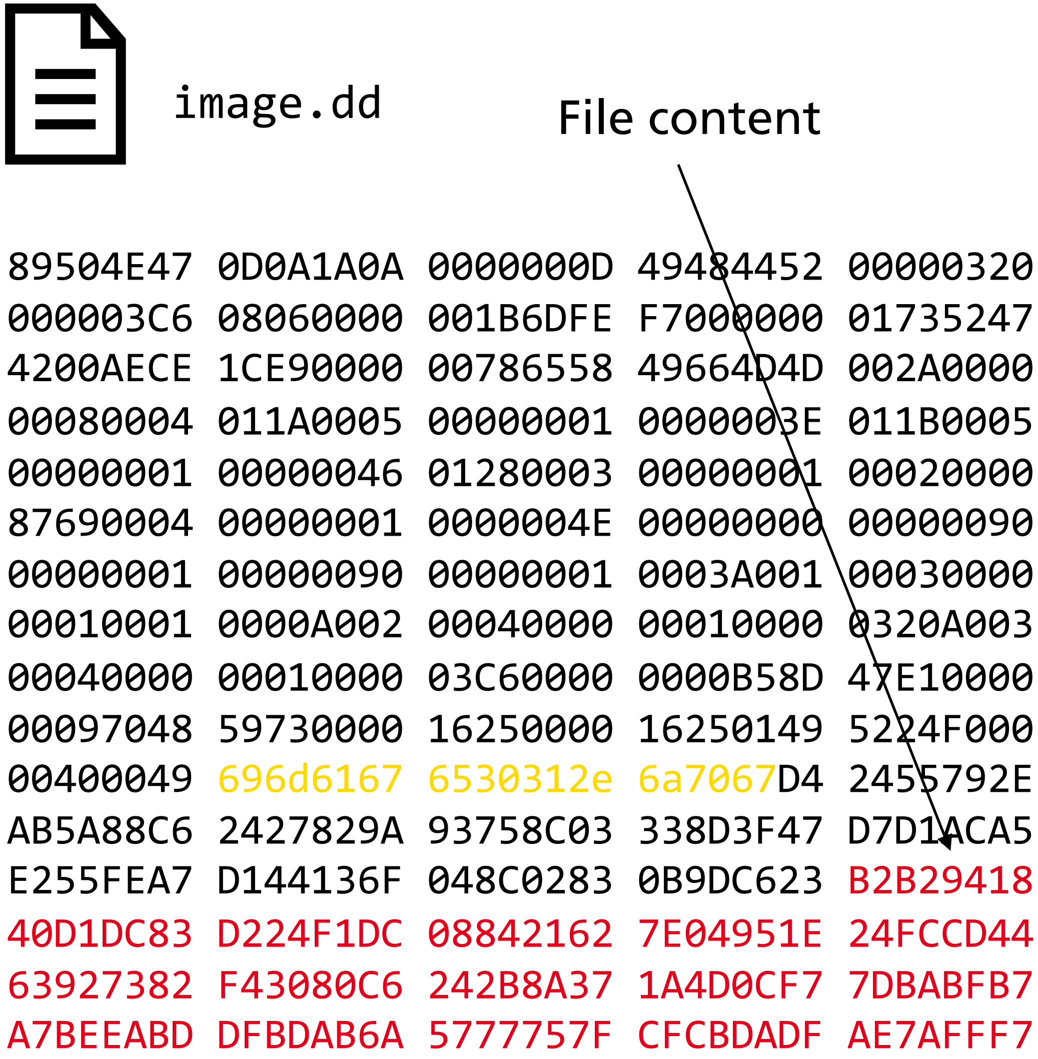
File name

```
89504E47 0D0A1A0A 0000000D 49484452 00000320
000003C6 08060000 001B6DFE F7000000 01735247
4200AECE 1CE90000 00786558 4964D4D 002A0000
00080004 011A0005 00000001 0000003E 011B0005
00000001 00000046 01280003 00000001 00020000
87690004 00000001 0000004E 00000000 00000090
00000001 00000090 00000001 0003A001 00030000
00010001 0000A002 00040000 00010000 0320A003
00040000 00010000 03C60000 0000B58D 47E10000
00097048 59730000 16250000 16250149 5224F000
00400049 696d6167 6530312e 6a7067D4 2455792E
AB5A88C6 2427829A 93758C03 338D3F47 D7D1ACA5
E255FEA7 D144136F 048C0283 0B9DC623 B2B29418
40D1DC83 D224F1DC 08842162 7E04951E 24FCCD44
63927382 F43080C6 242B8A37 1A4D0CF7 7DBABFB7
A7BEEABD DFBDAB6A 5777757F CFCBDADF AE7AFFF7
```

File Systems

„*The motivation behind a file system is fairly simple: computers need a method for the long-term storage and retrieval of data. File systems provide a mechanism for users to store data in a hierarchy of files and directories. A file system consists of structural and user data that are organized such that the computer knows where to find them. In most cases, the file system is independent from any specific computer.*“

Dr. Brian Carrier



```
89504E47 0D0A1A0A 0000000D 49484452 00000320
000003C6 08060000 001B6DFE F7000000 01735247
4200AECE 1CE90000 00786558 49664D4D 002A0000
00080004 011A0005 00000001 0000003E 011B0005
00000001 00000046 01280003 00000001 00020000
87690004 00000001 0000004E 00000000 00000090
00000001 00000090 00000001 0003A001 00030000
00010001 0000A002 00040000 00010000 0320A003
00040000 00010000 03C60000 0000B58D 47E10000
00097048 59730000 16250000 16250149 5224F000
00400049 696d6167 6530312e 6a7067D4 2455792E
AB5A88C6 2427829A 93758C03 338D3F47 D7D1ACA5
E255FEA7 D144136F 048C0283 0B9DC623 B2B29418
40D1DC83 D224F1DC 08842162 7E04951E 24FCCD44
63927382 F43080C6 242B8A37 1A4D0CF7 7DBABFB7
A7BEEABD DFBDAB6A 5777757F CFCBDADF AE7AFFF7
```

File Systems

„The motivation behind a file system is fairly simple: computers need a method for the long-term storage and retrieval of data. File systems provide a mechanism for users to store data in a hierarchy of files and directories. A file system consists of structural and user data that are organized such that the computer knows where to find them. In most cases, the file system is independent from any specific computer."

Dr. Brian Carrier

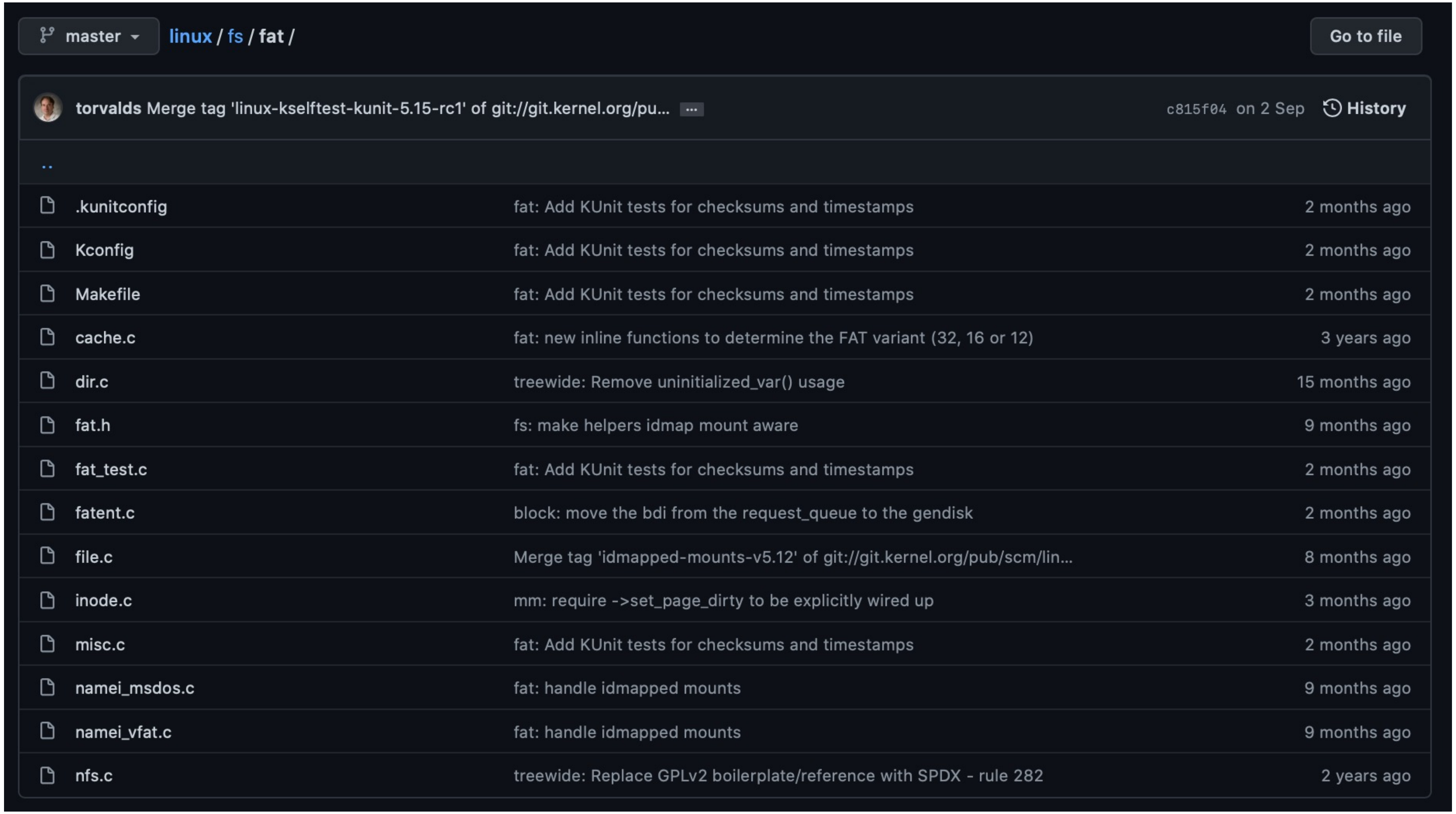
Operating systems need to be able to parse and understand all of this file system information!

image.dd

Timestamp

```
89504E47 0D0A1A0A 0000000D 49484452 00000320
000003C6 08060000 001B6DFE F7000000 01735247
4200AECE 1CE90000 00786558 49664D4D 002A0000
00080004 011A0005 00000001 0000003E 011B0005
00000001 00000046 01280003 00000001 00020000
87690004 00000001 0000004E 00000000 00000090
00000001 00000090 00000001 0003A001 00030000
00010001 0000A002 00040000 00010000 0320A003
00040000 00010000 03C60000 0000B58D 47E10000
00097048 59730000 16250000 16250149 5224F000
00400049 696d6167 6530312e 6a7067D4 2455792E
AB5A88C6 2427829A 93758C03 338D3F47 D7D1ACA5
E255FEA7 D144136F 048C0283 0B9DC623 B2B29418
40D1DC83 D224F1DC 08842162 7E04951E 24FCCD44
63927382 F43080C6 242B8A37 1A4D0CF7 7DBABFB7
A7BEEABD DFBDAB6A 5777757F CFCBDADF AE7AFFF7
```

File Systems



linux / fs / fat /

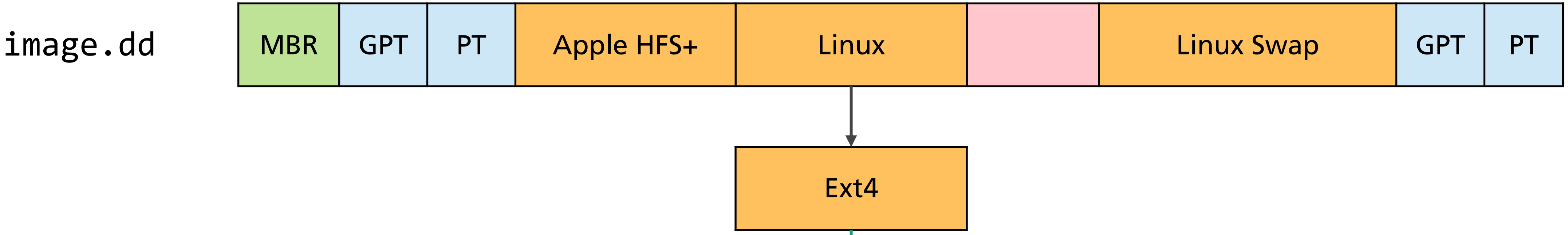
torvalds Merge tag 'linux-kselftest-kunit-5.15-rc1' of git://git.kernel.org/pu... c815f04 on 2 Sep

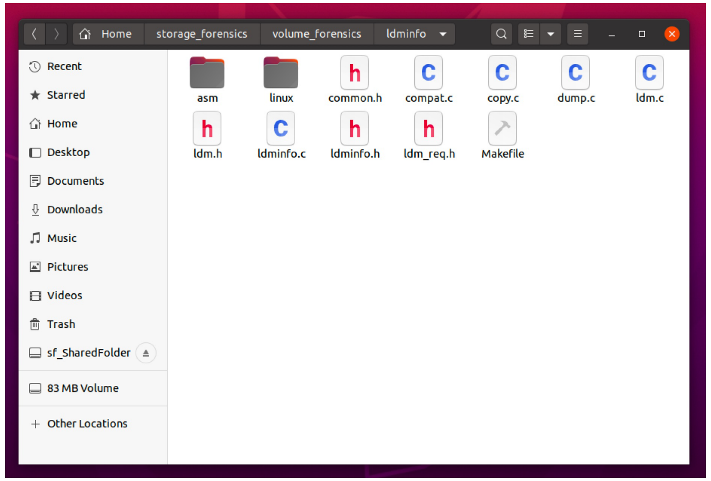
..

.kunitconfig	fat: Add KUnit tests for checksums and timestamps	2 months ago
Kconfig	fat: Add KUnit tests for checksums and timestamps	2 months ago
Makefile	fat: Add KUnit tests for checksums and timestamps	2 months ago
cache.c	fat: new inline functions to determine the FAT variant (32, 16 or 12)	3 years ago
dir.c	treewide: Remove uninitialized_var() usage	15 months ago
fat.h	fs: make helpers idmap mount aware	9 months ago
fat_test.c	fat: Add KUnit tests for checksums and timestamps	2 months ago
fatent.c	block: move the bdi from the request_queue to the gendisk	2 months ago
file.c	Merge tag 'idmapped-mounts-v5.12' of git://git.kernel.org/pub/scm/lin...	8 months ago
inode.c	mm: require ->set_page_dirty to be explicitly wired up	3 months ago
misc.c	fat: Add KUnit tests for checksums and timestamps	2 months ago
namei_msdos.c	fat: handle idmapped mounts	9 months ago
namei_vfat.c	fat: handle idmapped mounts	9 months ago
nfs.c	treewide: Replace GPLv2 boilerplate/reference with SPDX - rule 282	2 years ago

https://github.com/torvalds/linux/tree/master/fs

File System Analysis





Mount a file system to the system's own virtual file system in order to access its files.

File System Analysis

mount

mount(8) - Linux man page

Name

mount - mount a filesystem

Synopsis

mount [**-lhV**]

mount -a [**-fFnrsvw**] [**-t** *vfstype*] [**-O** *optlist*]

mount [**-fnrsvw**] [**-o** *option*[,*option*]...] *device*|*dir*

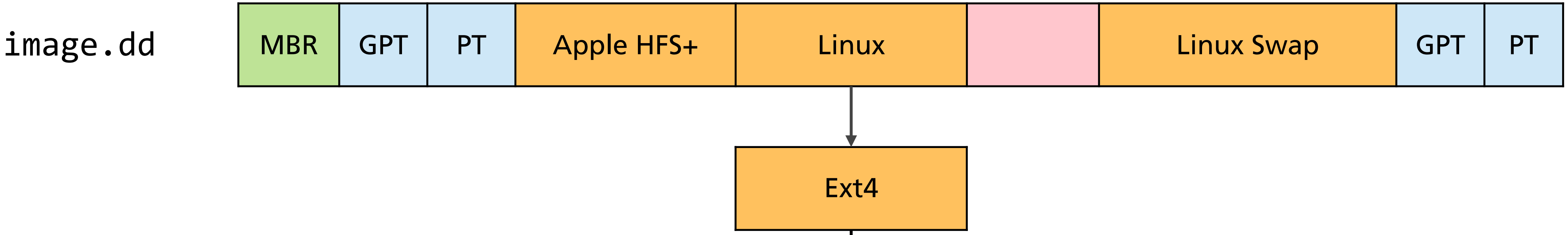
mount [**-fnrsvw**] [**-t** *vfstype*] [**-o** *options*] *device dir*

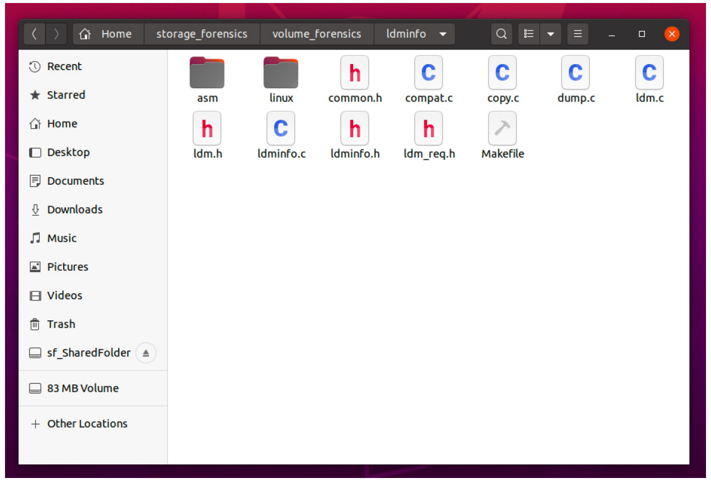
Description

All files accessible in a Unix system anged in one big tree, the file
hierarchy, rooted at /. These files ead out over several devices.
The **mount** command serves to attach t tem found on some device to the big
file tree. Conversely, the **umount**(8 ll detach it again.

Mount has a -o ro (for read-only) option!

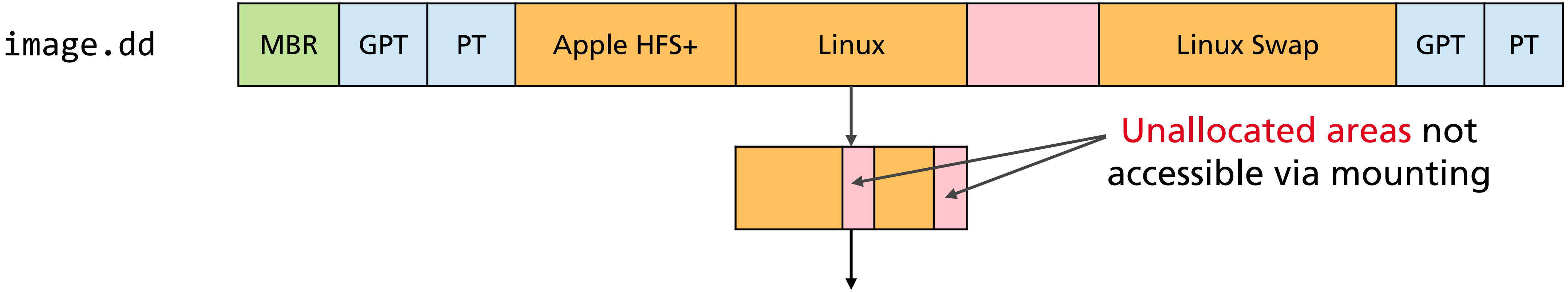
File System Analysis



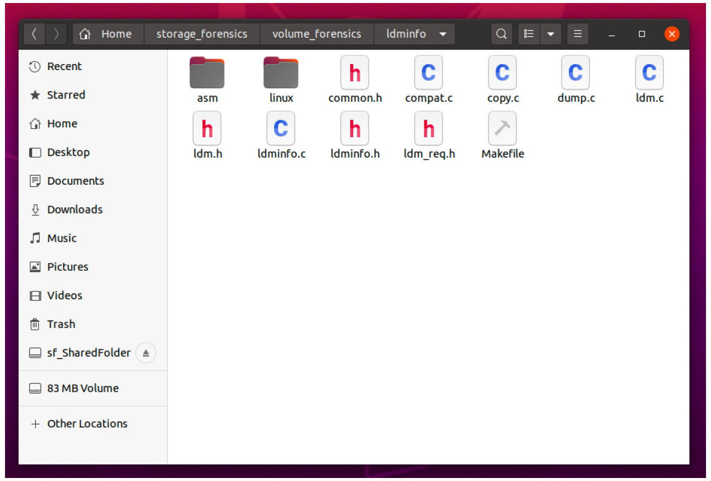


- Analysis depends on the used implementation
- Not all available information may be accessible

File System Analysis



Unallocated areas not accessible via mounting



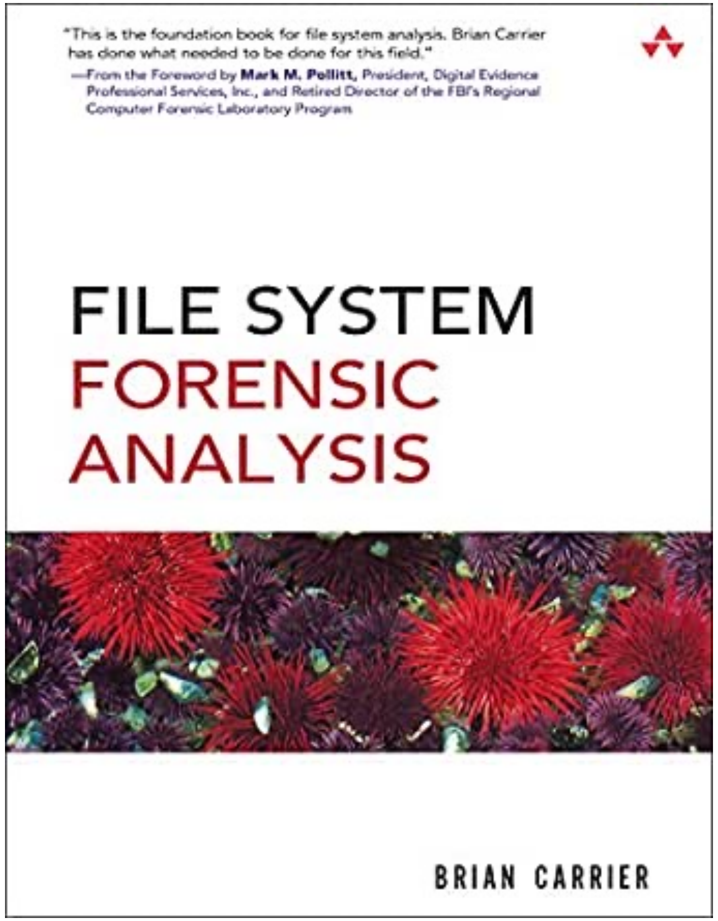
- Analysis depends on the used implementation
- Not all available information may be accessible
- Deleted files are not mounted

Use an independent analysis tool

Make sure all important information is easily accessible

Perform the analysis on the whole volume

The Sleuth Kit (TSK)



"This is the foundation book for file system analysis. Brian Carrier has done what needed to be done for this field."
—From the Foreword by **Mark M. Pollitt**, President, Digital Evidence Professional Services, Inc., and Retired Director of the FBI's Regional Computer Forensic Laboratory Program

FILE SYSTEM FORENSIC ANALYSIS

BRIAN CARRIER

Of course, TSK is not the only tool which performs file system analysis.
But it's free, well-known and widely used!

Open Source Digital Forensics

Autopsy® is an easy to use, GUI-based program that allows you to efficiently analyze hard drives and smart phones. It has a plug-in architecture that allows you to find add-on modules or develop custom modules in Java or Python.

The Sleuth Kit® is a collection of command line tools and a C library that allows you to analyze disk images and recover files from them. It is used behind the scenes in Autopsy and many other open source and commercial forensics tools.

These tools are used by thousands of users around the world and have community-based e-mail lists and forums. Commercial training, support, and custom development is available from Basis Technology.

https://www.sleuthkit.org

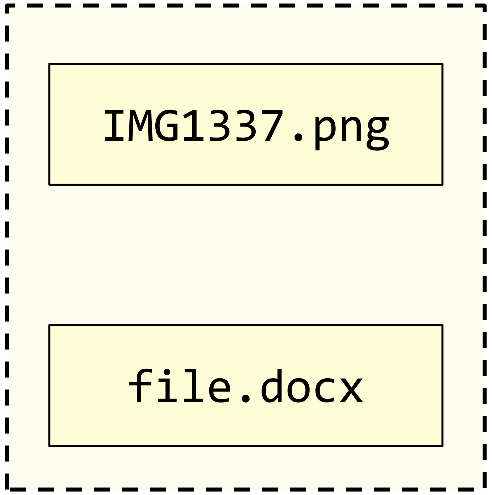
File Systems

File Name Category

- Enables users to refer to a file using a name instead of an address
- Stores usually only a name and points to a metadata entry

1. How to reference a file?

File Name Category

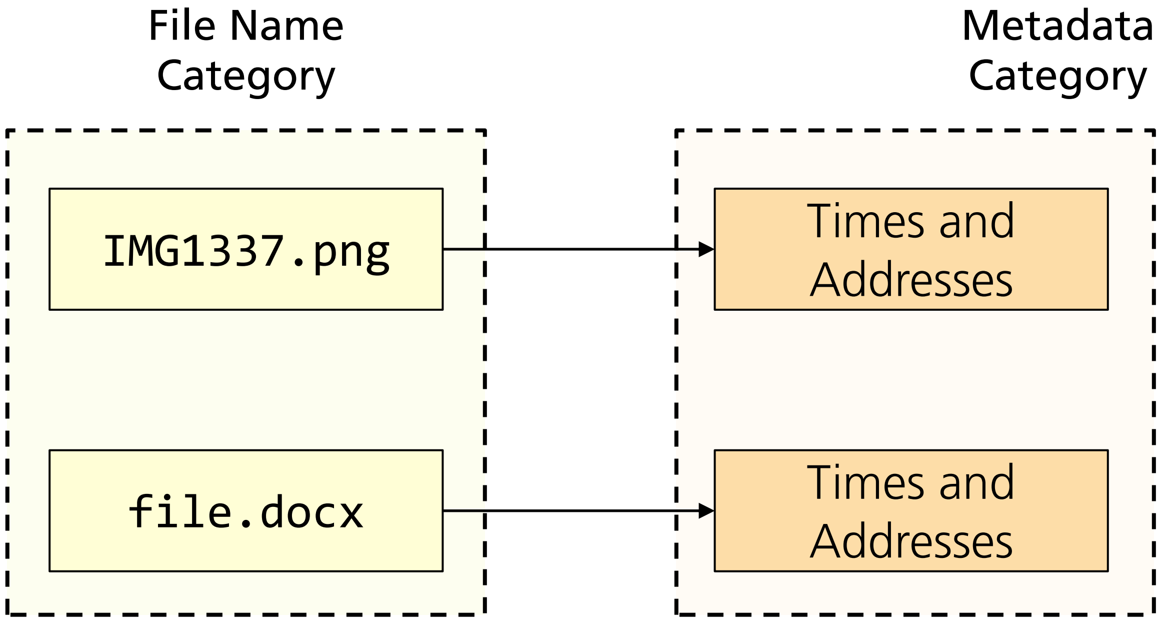


File Systems

Metadata Category

2. How to quickly find the start of a file and collect metadata (such as timestamps)?

- Stores metadata about a file e.g. timestamps, sizes or attributes
- Often implemented by metadata entries with their own allocation status
- Often file name and metadata data are stored in one place
- Points to the actual content of a file

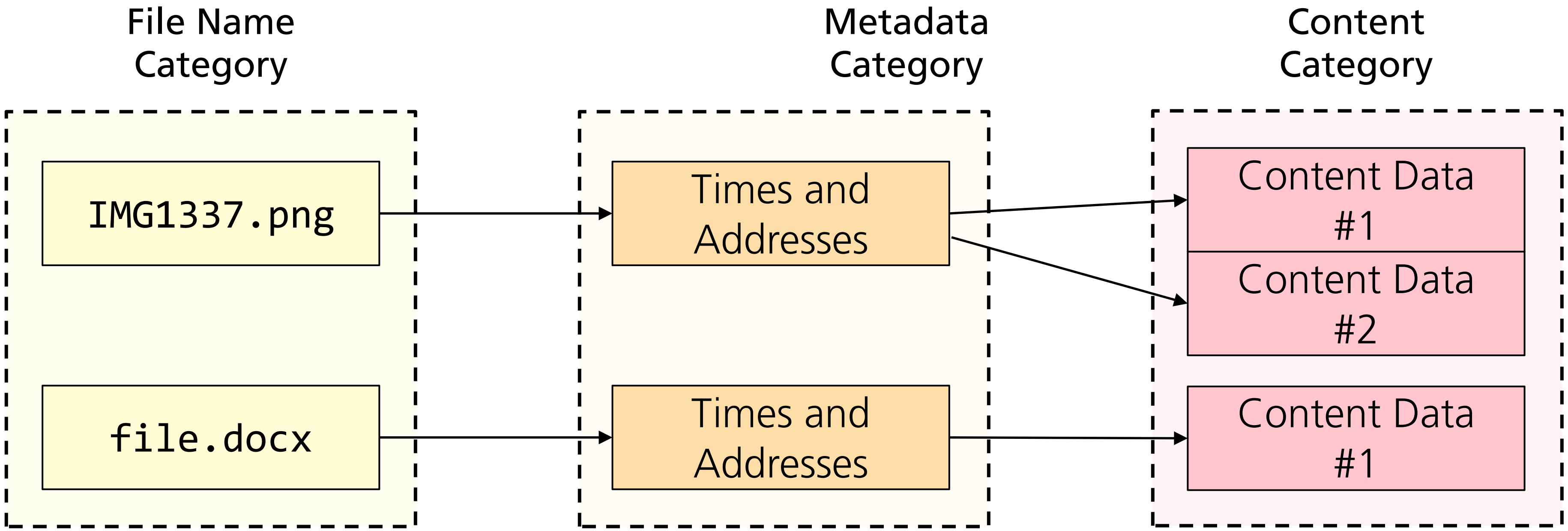


File Systems

Content Category

3. How to organize available space?

- Stores the contents of a file or directory
- Storage space is usually divided into data units (e.g. clusters or blocks) with their own allocation status
- It is sometimes not possible to store data units in consecutive order, which leads to fragmentation

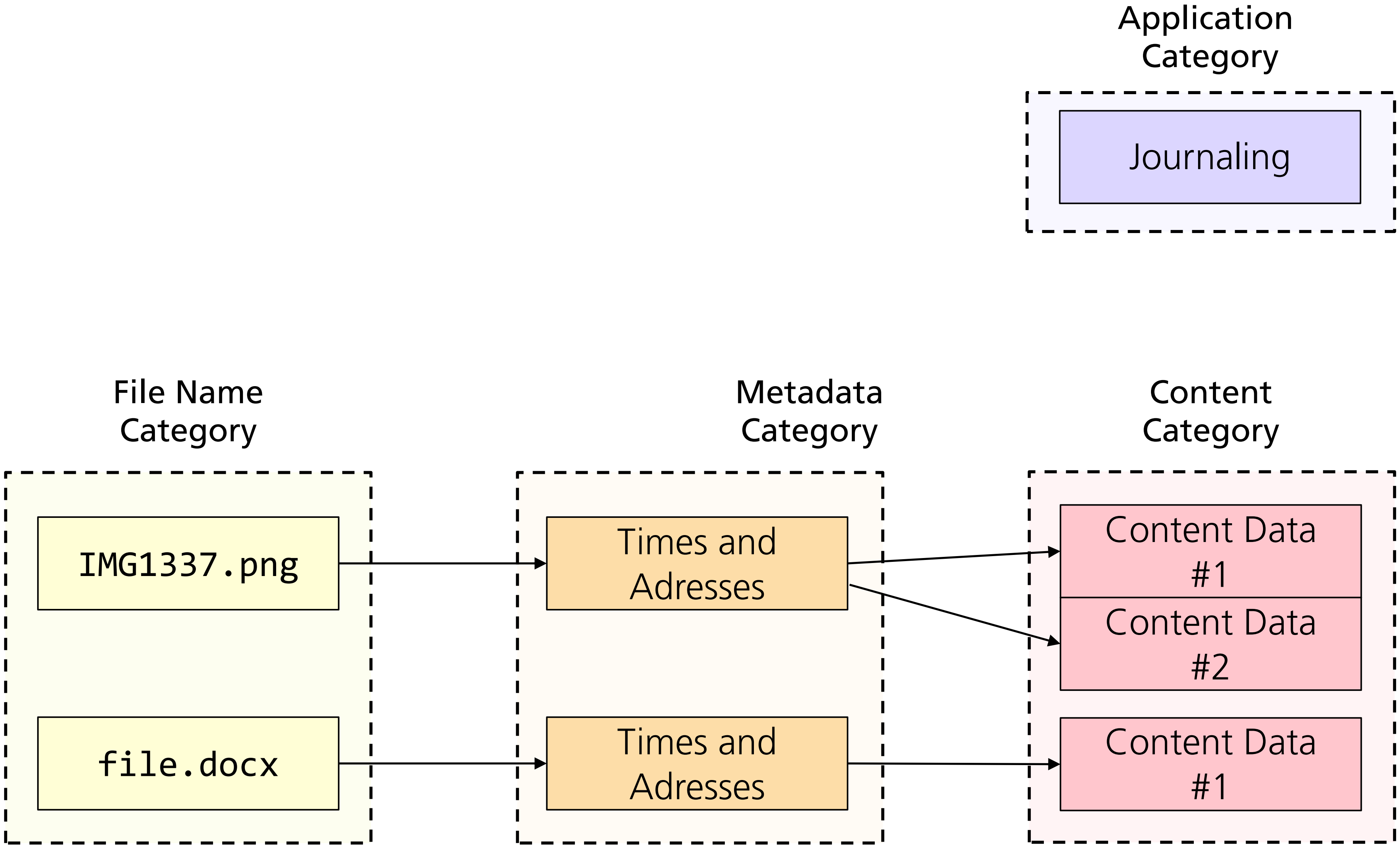


File Systems

Application Category

4. How to implement features such as logging?

- Includes non-essential data used for special file system features, e.g. journaling or a quota

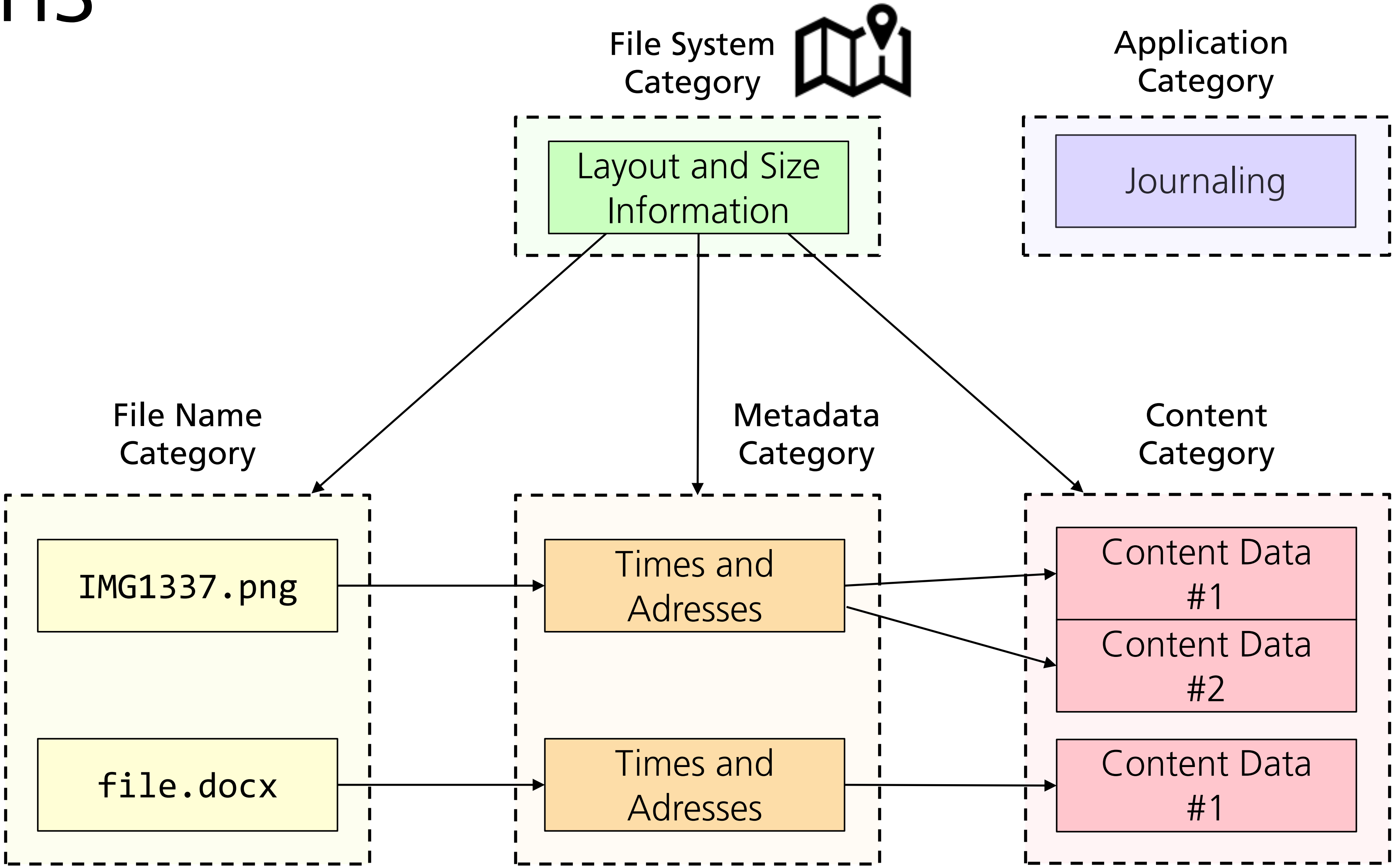


File Systems

File System Category

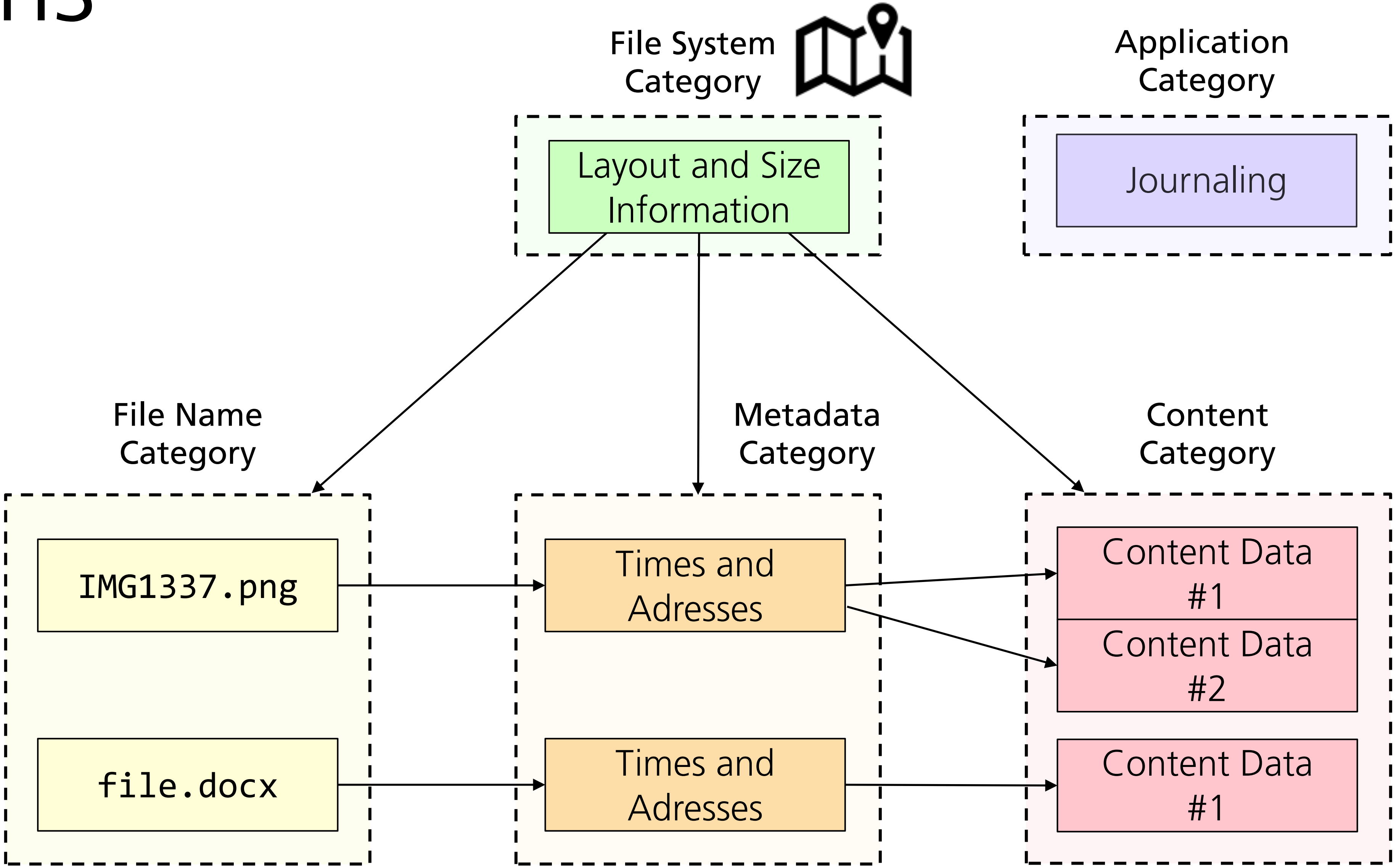
- Contains information about the layout of a file system including the location and sizes of its important structures
- Includes metadata about a file system (e.g. timestamps)
- Crucial data for further analysis of a file system

5. How to organize the aforementioned concepts?

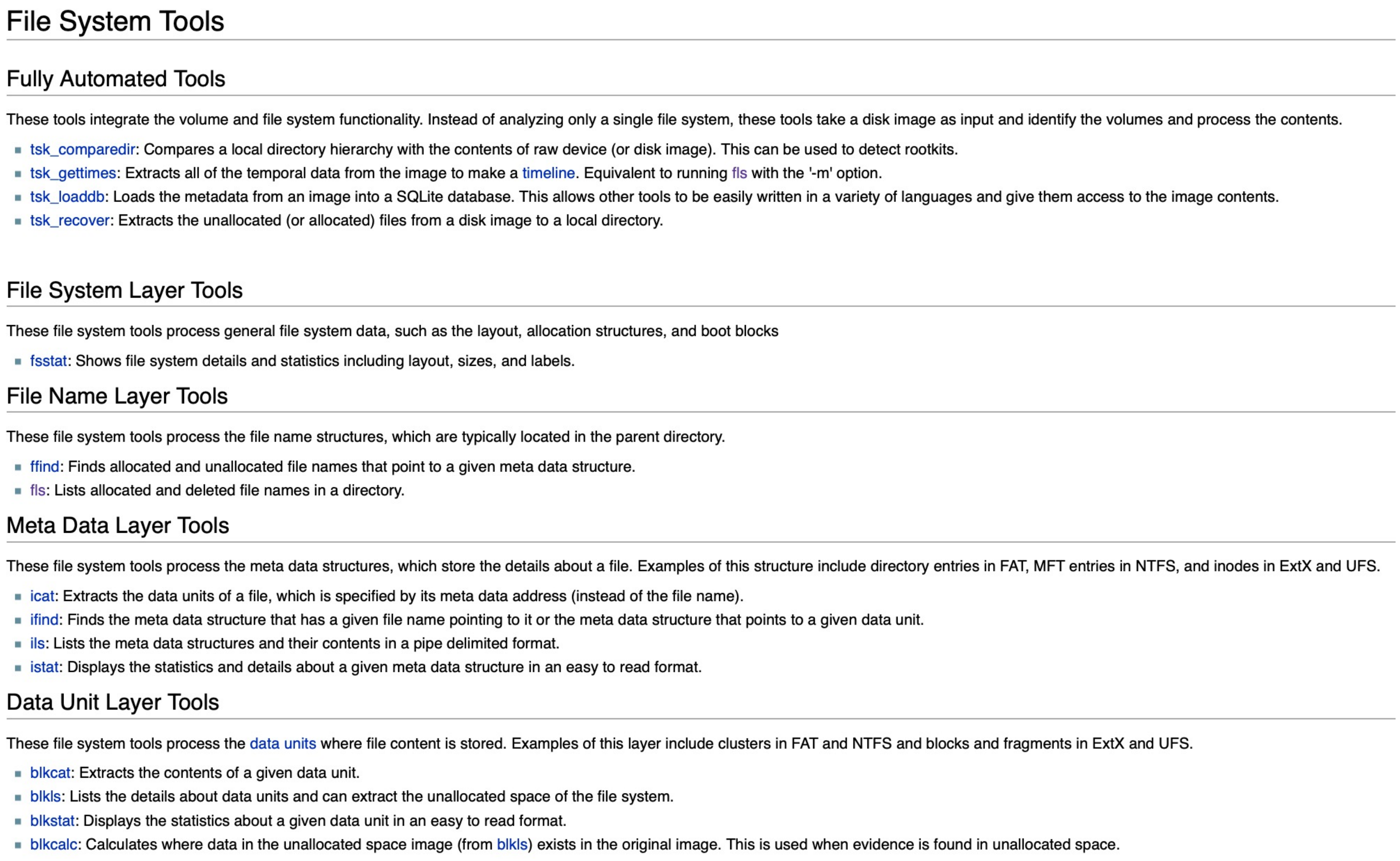


File Systems

Dr. Brian Carrier



The Sleuth Kit



File System Tools

Fully Automated Tools

These tools integrate the volume and file system functionality. Instead of analyzing only a single file system, these tools take a disk image as input and identify the volumes and process the contents.

- tsk_comparedir: Compares a local directory hierarchy with the contents of raw device (or disk image). This can be used to detect rootkits.
- tsk_gettimes: Extracts all of the temporal data from the image to make a timeline. Equivalent to running fls with the '-m' option.
- tsk_loaddb: Loads the metadata from an image into a SQLite database. This allows other tools to be easily written in a variety of languages and give them access to the image contents.
- tsk_recover: Extracts the unallocated (or allocated) files from a disk image to a local directory.

File System Layer Tools

These file system tools process general file system data, such as the layout, allocation structures, and boot blocks

- fsstat: Shows file system details and statistics including layout, sizes, and labels.

File Name Layer Tools

These file system tools process the file name structures, which are typically located in the parent directory.

- ffind: Finds allocated and unallocated file names that point to a given meta data structure.
- fls: Lists allocated and deleted file names in a directory.

Meta Data Layer Tools

These file system tools process the meta data structures, which store the details about a file. Examples of this structure include directory entries in FAT, MFT entries in NTFS, and inodes in ExtX and UFS.

- icat: Extracts the data units of a file, which is specified by its meta data address (instead of the file name).
- ifind: Finds the meta data structure that has a given file name pointing to it or the meta data structure that points to a given data unit.
- ils: Lists the meta data structures and their contents in a pipe delimited format.
- istat: Displays the statistics and details about a given meta data structure in an easy to read format.

Data Unit Layer Tools

These file system tools process the data units where file content is stored. Examples of this layer include clusters in FAT and NTFS and blocks and fragments in ExtX and UFS.

- blkcat: Extracts the contents of a given data unit.
- blkls: Lists the details about data units and can extract the unallocated space of the file system.
- blkstat: Displays the statistics about a given data unit in an easy to read format.
- blkcalc: Calculates where data in the unallocated space image (from blkls) exists in the original image. This is used when evidence is found in unallocated space.

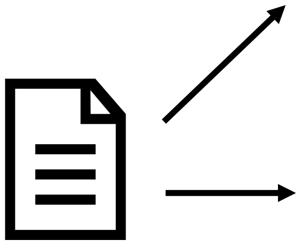
https://wiki.sleuthkit.org/index.php?title=TSK_Tool_Overview#File_System_Tools

File System Analysis

FAT File System

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10



Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

`file.docx`

Content

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

Metadata in FAT is stored in “Directory Entry” structures

`file.docx`

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00																
0x10																

Directory Entry

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

file.docx

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name															
0x10																

Directory Entry

Up to 11 characters of the file's name in ASCII padded with spaces

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

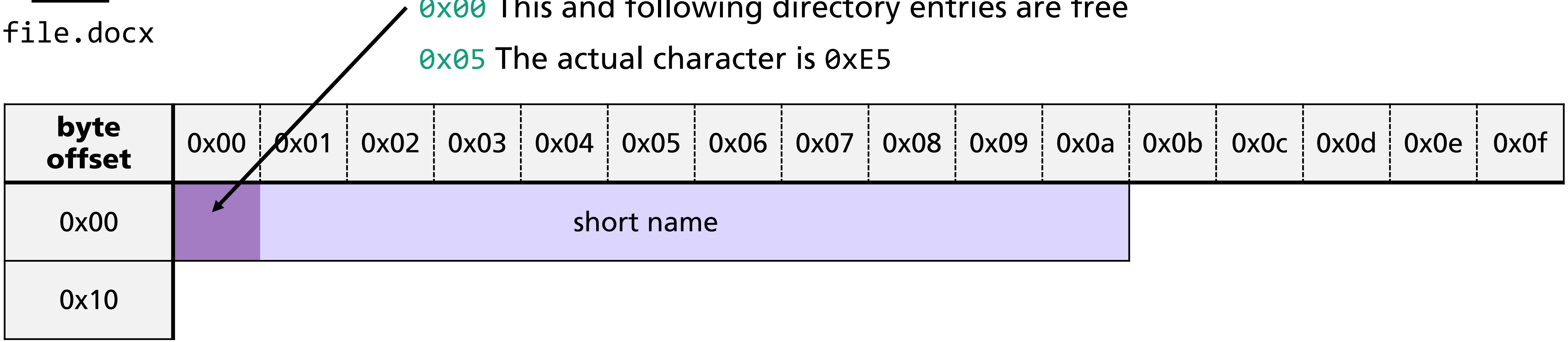
Metadata

file.docx

0xE5 This entry is free

0x00 This and following directory entries are free

0x05 The actual character is 0xE5



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00		short name														
0x10																

Directory Entry

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

`file.docx`

`0x01`	Read only
`0x02`	Hidden
`0x04`	Operating System file
`0x08`	Volume Label (should exist only onc
`0x10`	Directory
`0x20`	Archive

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.				
0x10																

Directory Entry

File attributes

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

file.docx

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.	res.	created time		
0x10	created date		date last accessed				time of last write		date of last write							

Directory Entry

FAT file system

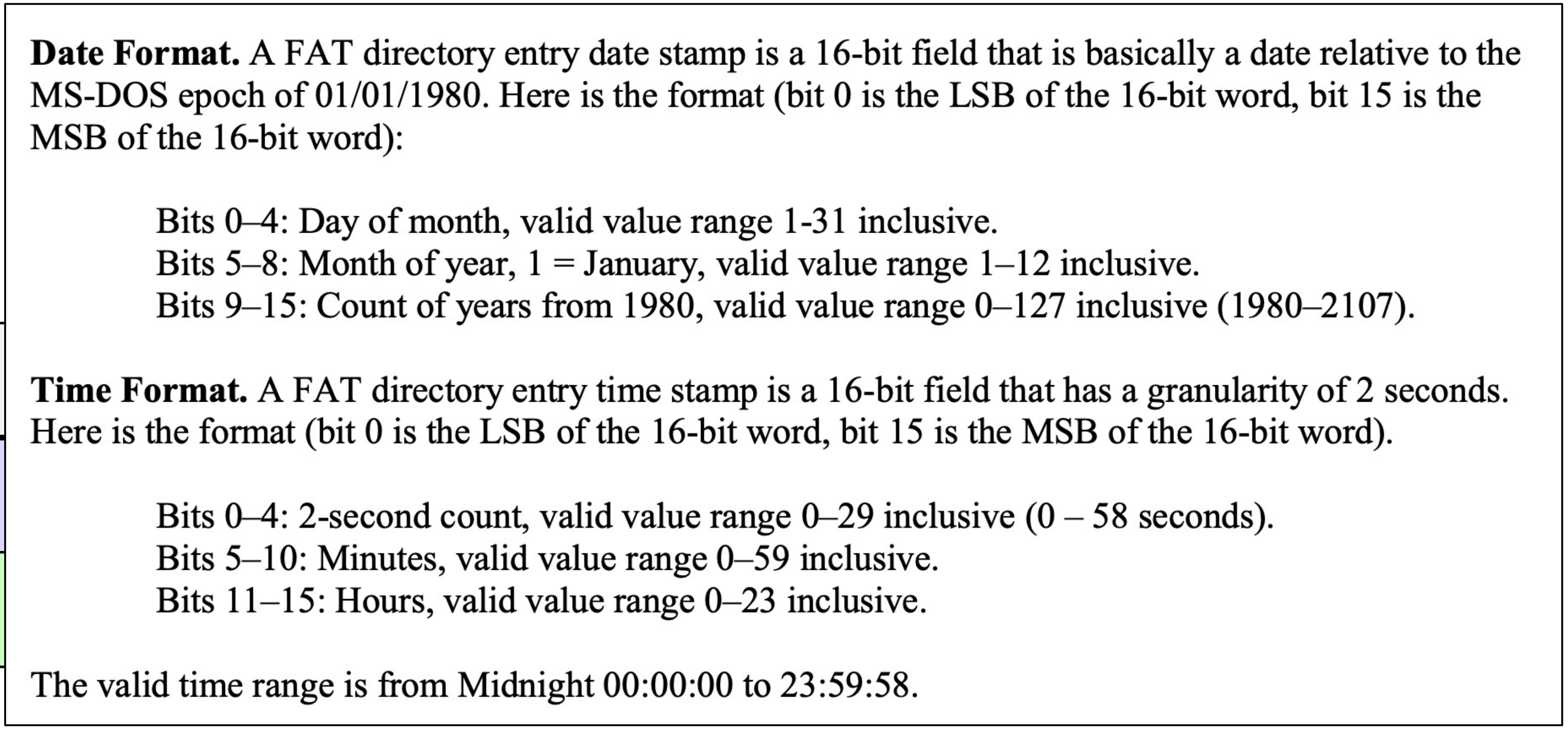
Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

`file.docx`

byte offset	0x00	0x01	0x02	0x03
0x00				
0x10	created date		date last accessed	

Directory Entry



Date Format. A FAT directory entry date stamp is a 16-bit field that is basically a date relative to the MS-DOS epoch of 01/01/1980. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word):

Bits 0–4: Day of month, valid value range 1-31 inclusive.
Bits 5–8: Month of year, 1 = January, valid value range 1–12 inclusive.
Bits 9–15: Count of years from 1980, valid value range 0–127 inclusive (1980–2107).

Time Format. A FAT directory entry time stamp is a 16-bit field that has a granularity of 2 seconds. Here is the format (bit 0 is the LSB of the 16-bit word, bit 15 is the MSB of the 16-bit word).

Bits 0–4: 2-second count, valid value range 0–29 inclusive (0 – 58 seconds).
Bits 5–10: Minutes, valid value range 0–59 inclusive.
Bits 11–15: Hours, valid value range 0–23 inclusive.

The valid time range is from Midnight 00:00:00 to 23:59:58.

`https://www.cs.virginia.edu/~cr4bd/4414/F2018/files/fatspec.pdf`

FAT file system

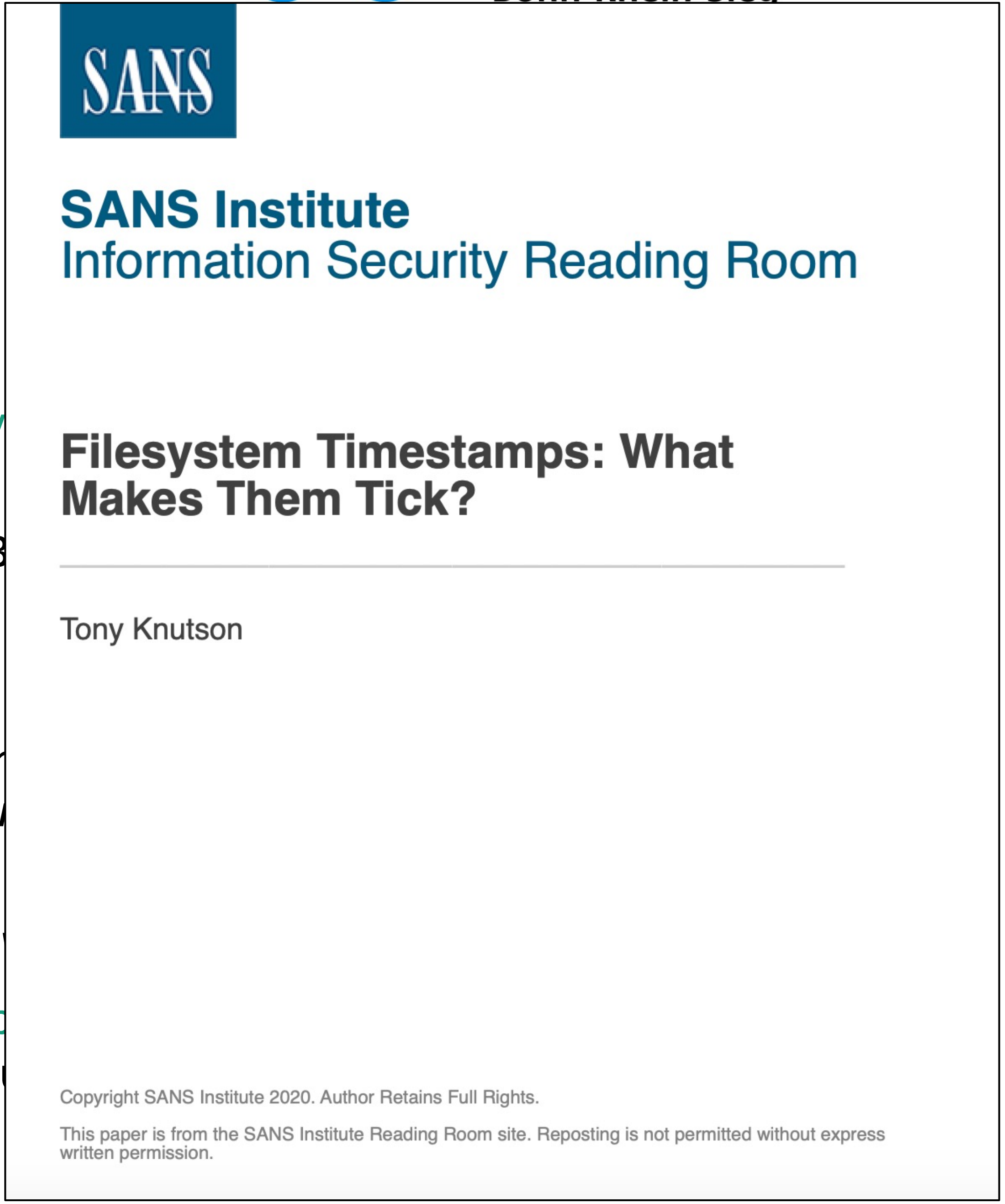
Analysis: Timestamps

- Timestamps provide valuable information during an investigation
- **However:** It is always important to know, what they mean i.e. when exactly they are updated
- **Example:** Findings for the Creation Time in FAT by Brian Carrier:
  - *„The created time is set when Windows allocates a new directory entry for a new file."*
  - *„[…] if the OS allocates a new directory entry for an existing file, even if original location was on a different disk, the original creation time is kept."*
  - *„There is one known exception to this rule, if the move is done from the command line of a 2000/XP system to a different volume."*
- **Summary:** Time Value Updating depends on a combination of file system and operating system (and tools used) and has to be evaluated in order to know the exact meaning of available time stamps

FAT file system

Analysis: Timestamps

- Timestamps provide valuable information during an
- **However:** It is always important to know, what they they are updated
- **Example:** Findings for the Creation Time in FAT by B
  - *„The created time is set when Windows allocates new file."*
  - *„[…] if the OS allocates a new directory entry for original location was on a different disk, the origi*
  - *„There is one known exception to this rule, if the command line of a 2000/XP system to a different*
- **Summary:** Time Value Updating depends on a comb operating system (and tools used) and has to be eval exact meaning of available time stamps



SANS

SANS Institute
Information Security Reading Room

Filesystem Timestamps: What Makes Them Tick?

Tony Knutson

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

`file.docx`

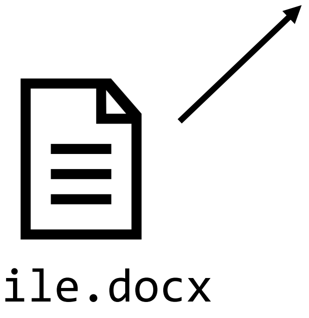
byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.	res.	created time		
0x10	created date		date last accessed				time of last write		date of last write							

Directory Entry

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata



`file.docx`

Number of first cluster
(high bytes are 0 for FAT12/16)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.	res.	created time		
0x10	created date		date last accessed		first cluster (high bytes)		time of last write		date of last write		first cluster (low bytes)					

Directory Entry

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

`file.docx`

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.	res.	created time		
0x10	created date		date last accessed		first cluster (high bytes)		time of last write		date of last write		first cluster (low bytes)		file size			

Directory Entry

File size in bytes (max. 4 GB!)

FAT file system

Name	Start	Size	Created on
file-with-a-long-name.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

What about „really long“ file names?

file*.docx

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.	res.	created time		
0x10	created date		date last accessed		first cluster (high bytes)		time of last write		date of last write		first cluster (low bytes)		file size			

Directory Entry

FAT file system

Name	Start	Size	Created on
`file-with-a-long-name.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

`file*.docx`

For longer names, the file name is split up and stored in multiple Long File Name (LFN) directory entries (in addition to a normal directory entry)!

<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td></td><td colspan="10">characters 1 – 5</td><td></td><td></td><td></td><td colspan="2"></td></tr>
<tr><td>0x10</td><td colspan="10">characters 6 – 11</td><td></td><td></td><td colspan="4">characters 12 – 13</td></tr>
</table>

LFN Directory Entry

FAT file system

Name	Start	Size	Created on
`file-with-a-long-name.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

`file*.docx`

Used to order multiple LFN entries.
Last part is masked with `0x40`
and should be the first LFN entry.

Sequence #	Entry data
`0x42`	`ong-name.docx`
`0x01`	`file-with-a-l`
-	Default Directory Entry

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	seq. #	characters 1 – 5														
0x10	characters 6 – 11												characters 12 – 13			

LFN Directory Entry

FAT file system

Name	Start	Size	Created on
file-with-a-long-name.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

file*.docx

Zero indicates an entry that is part of a long file name

Special attribute value

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	seq. #	characters 1 – 5										0x0F	0x00	check-sum		
0x10	characters 6 – 11												characters 12 – 13			

LFN Directory Entry

FAT file system

Name	Start	Size	Created on
file-with-a-long-name.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

file*.docx

Checksum of the corresponding short file name

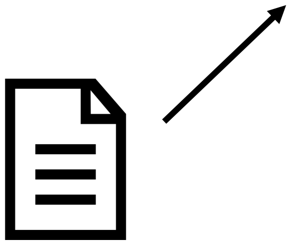
byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	seq. #	characters 1 – 5										0x0F	0x00	check-sum		
0x10	characters 6 – 11												characters 12 – 13			

LFN Directory Entry

FAT file system

Name	Start	Size	Created on
file-with-a-long-name.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata



file*.docx

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	seq. #	characters 1 – 5										0x0F	0x00	check-sum		
0x10	characters 6 – 11										0x0000		characters 12 – 13			

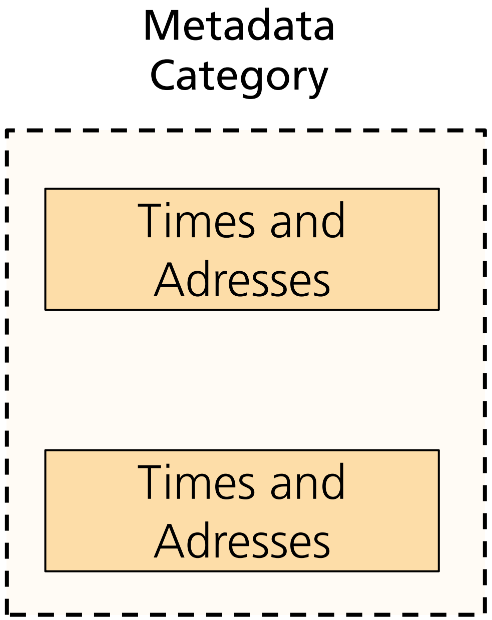
LFN Directory Entry

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

file.docx



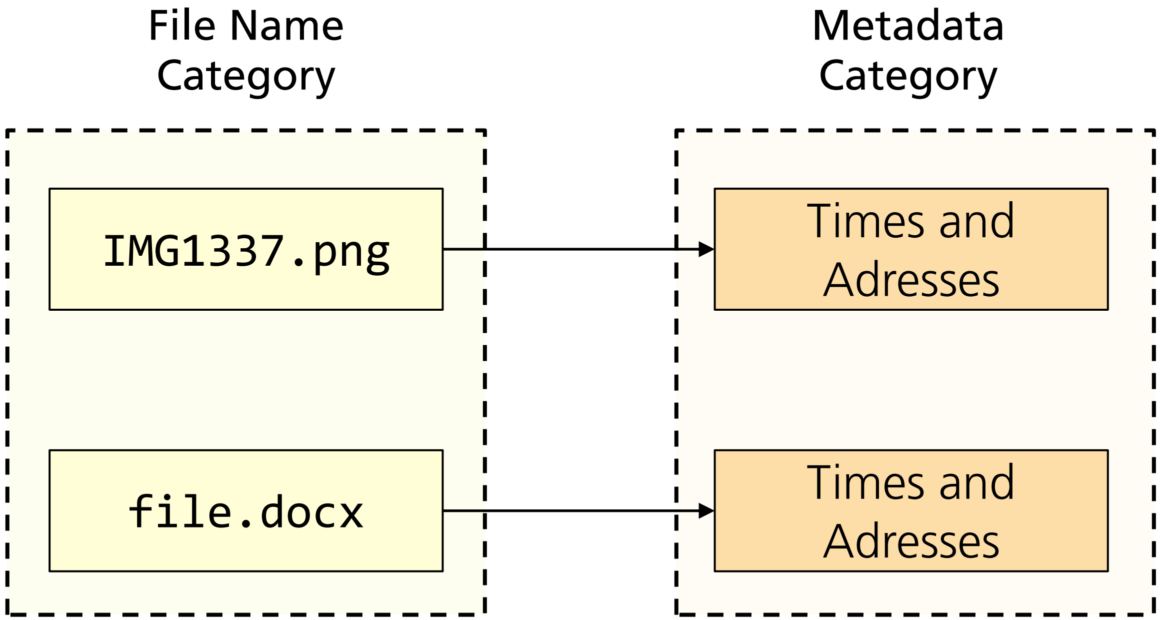
Directory entries belong to the metadata category…

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

file.docx



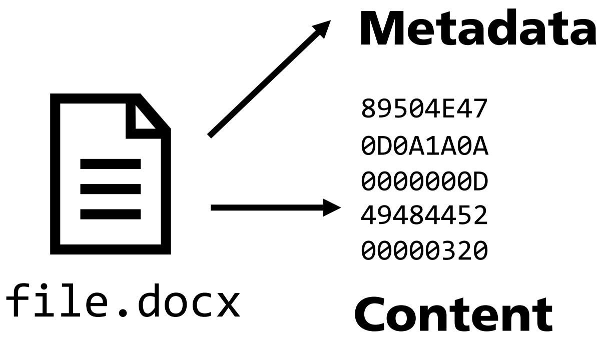
Directory entries belong to the metadata category…

…but also partially contain file name data.

LFNs are pure file name data

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

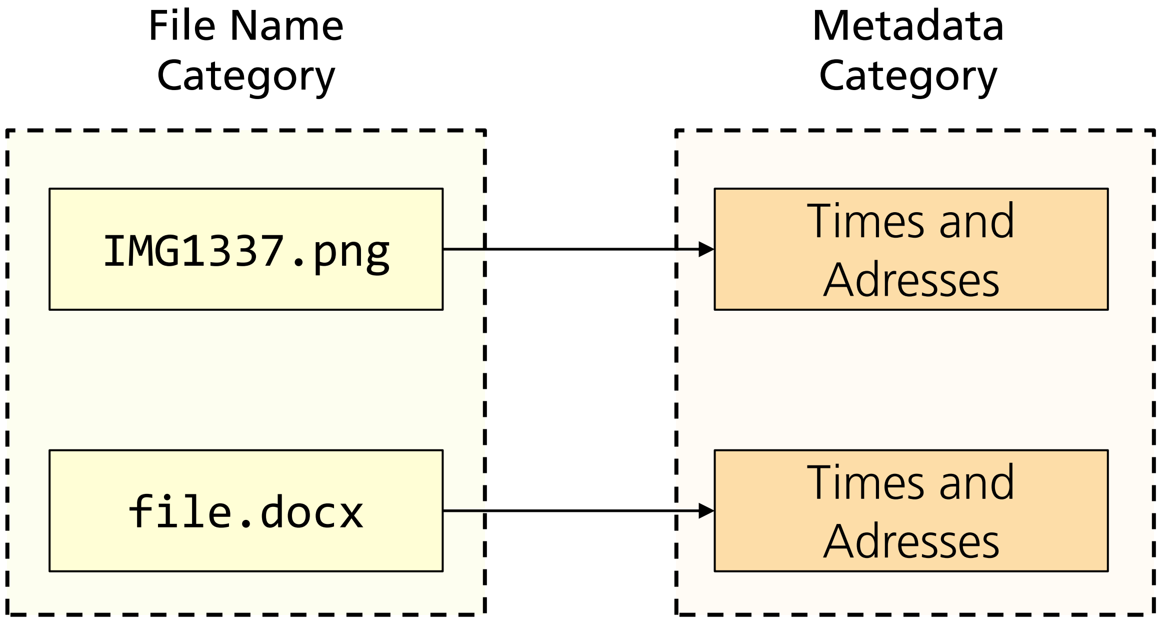


file.docx

Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content



File Name Category

Metadata Category

IMG1337.png

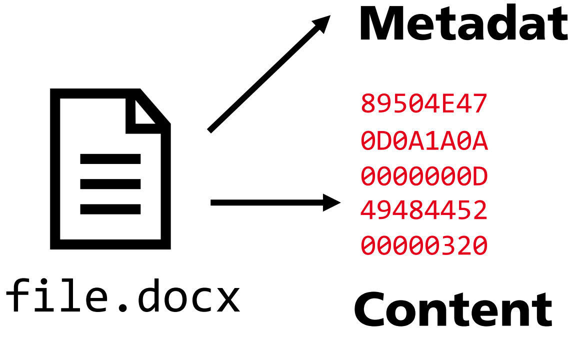
file.docx

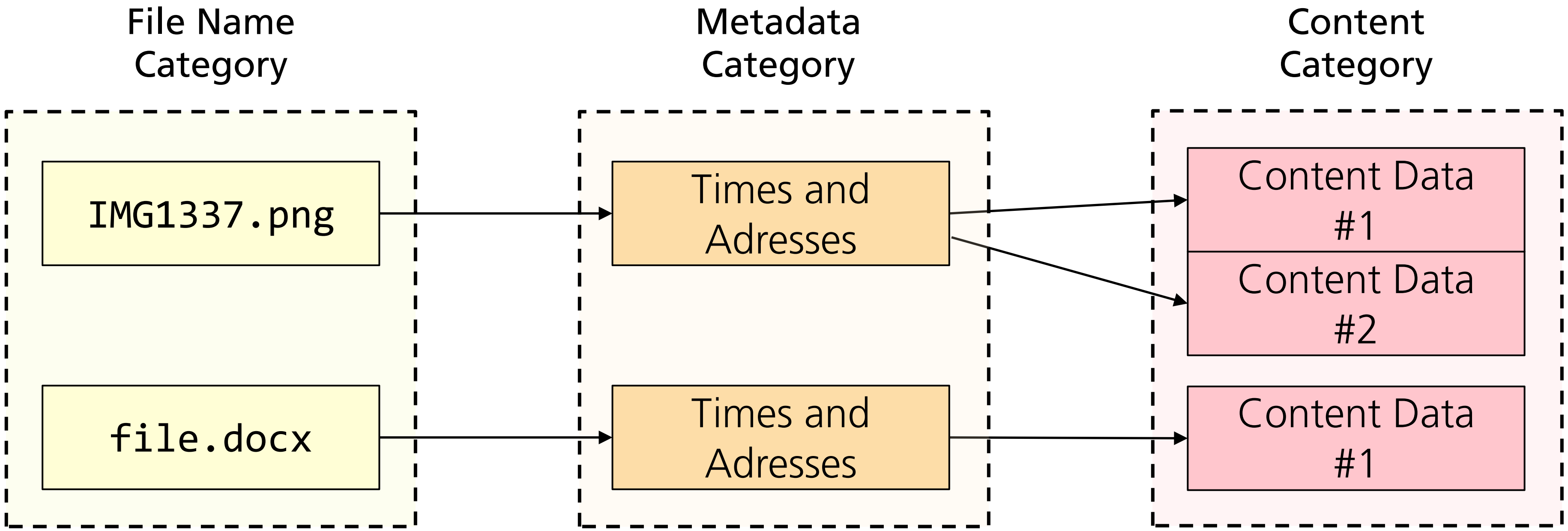
Times and Adresses

Times and Adresses

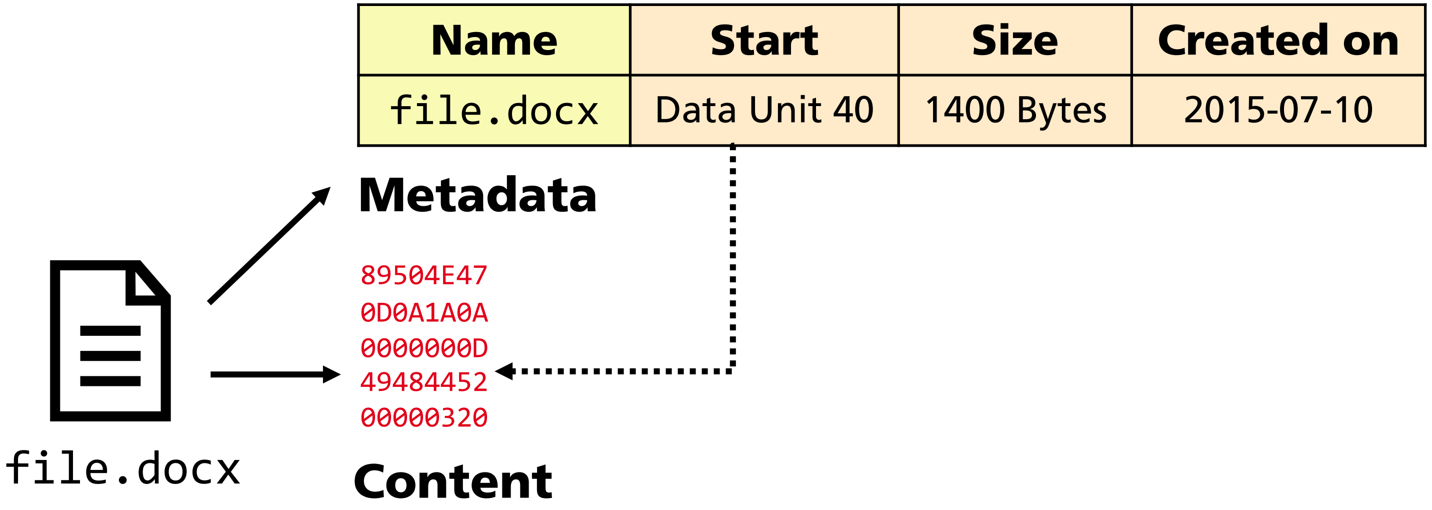
FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10





FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

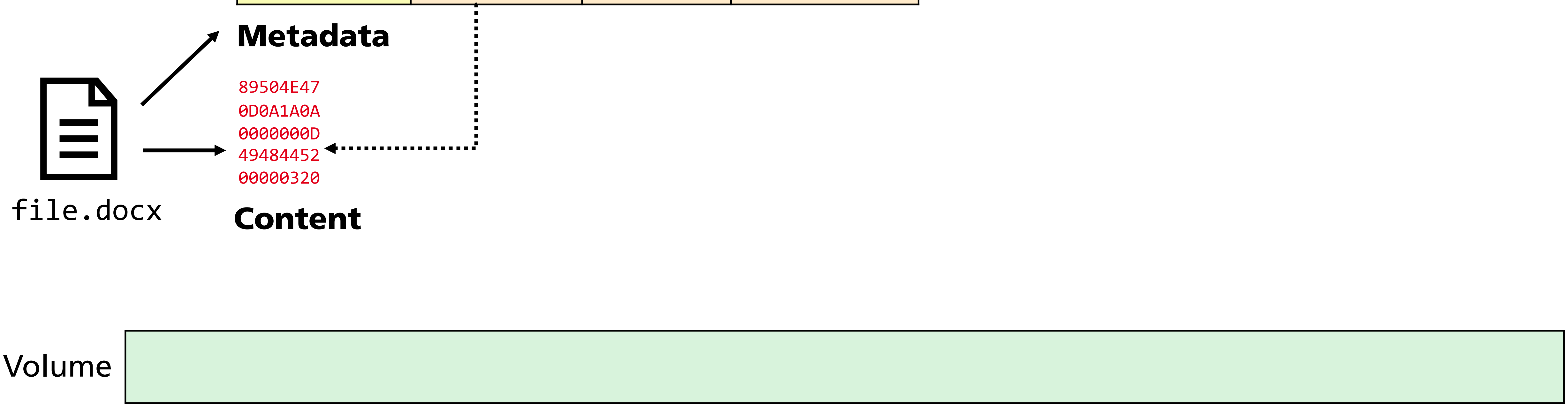
Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

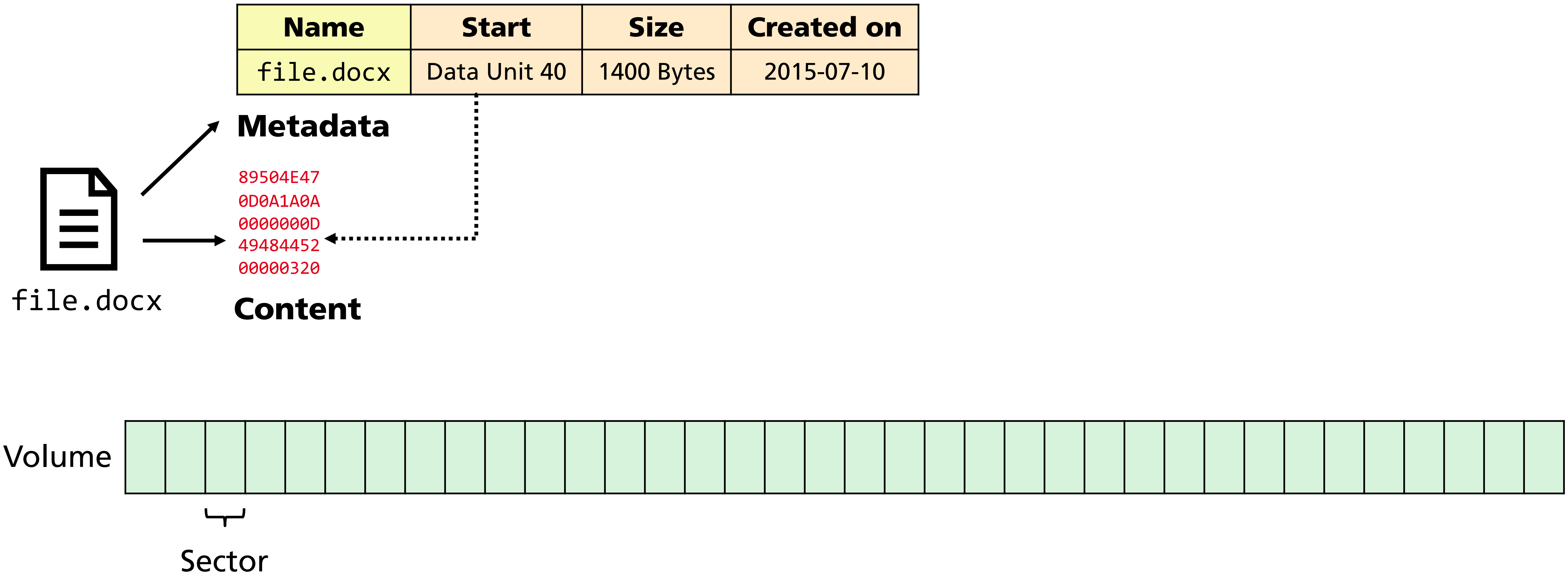
FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10



```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

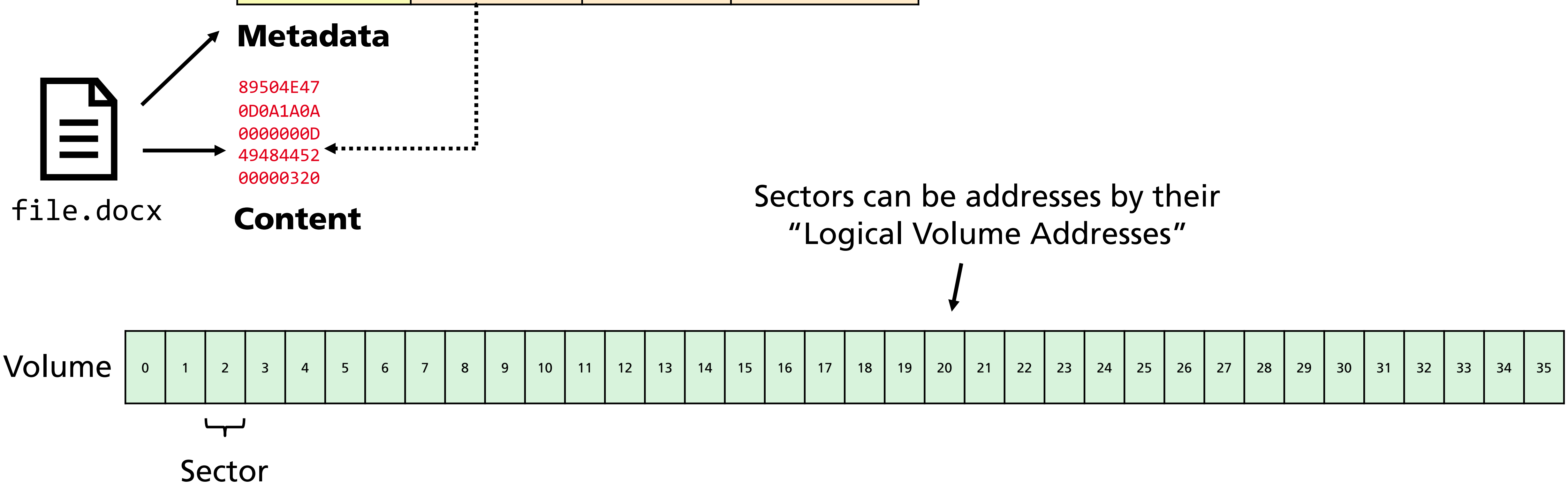
Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10



Metadata

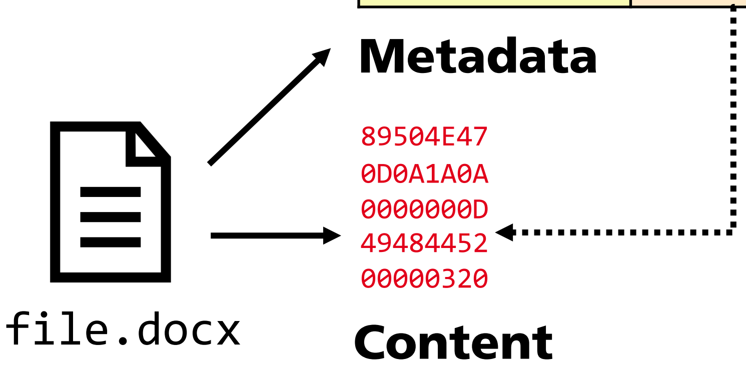
```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

Sectors can be addresses by their “Logical Volume Addresses”

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

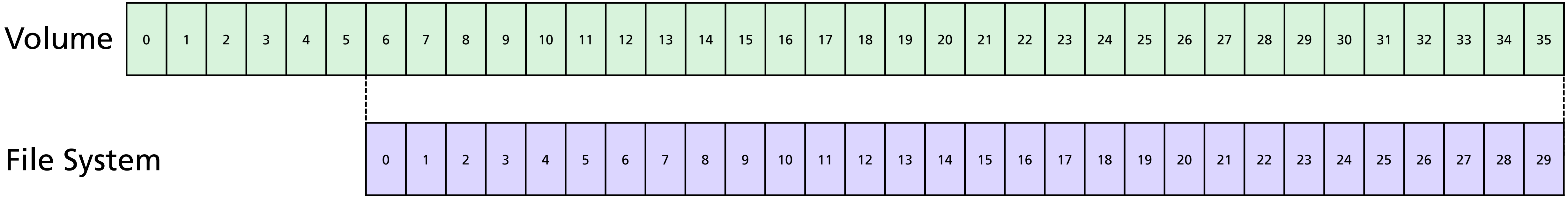


file.docx

Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

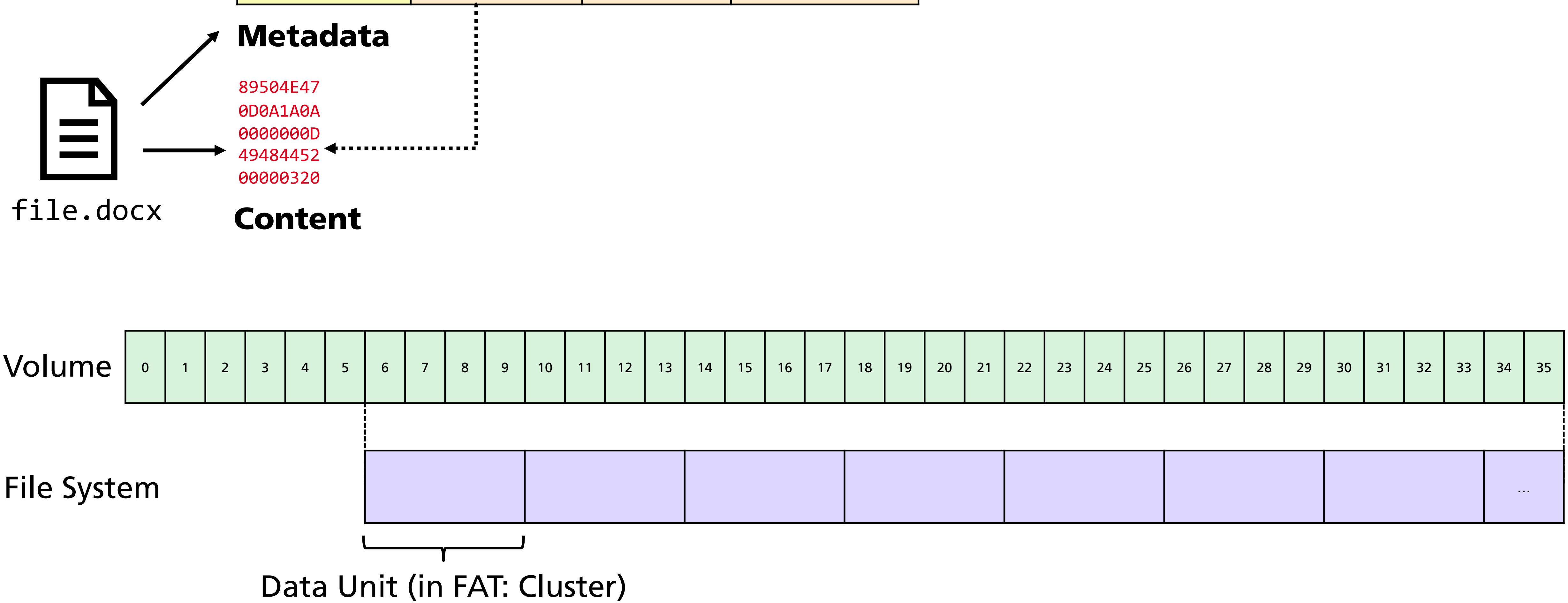


Volume: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35

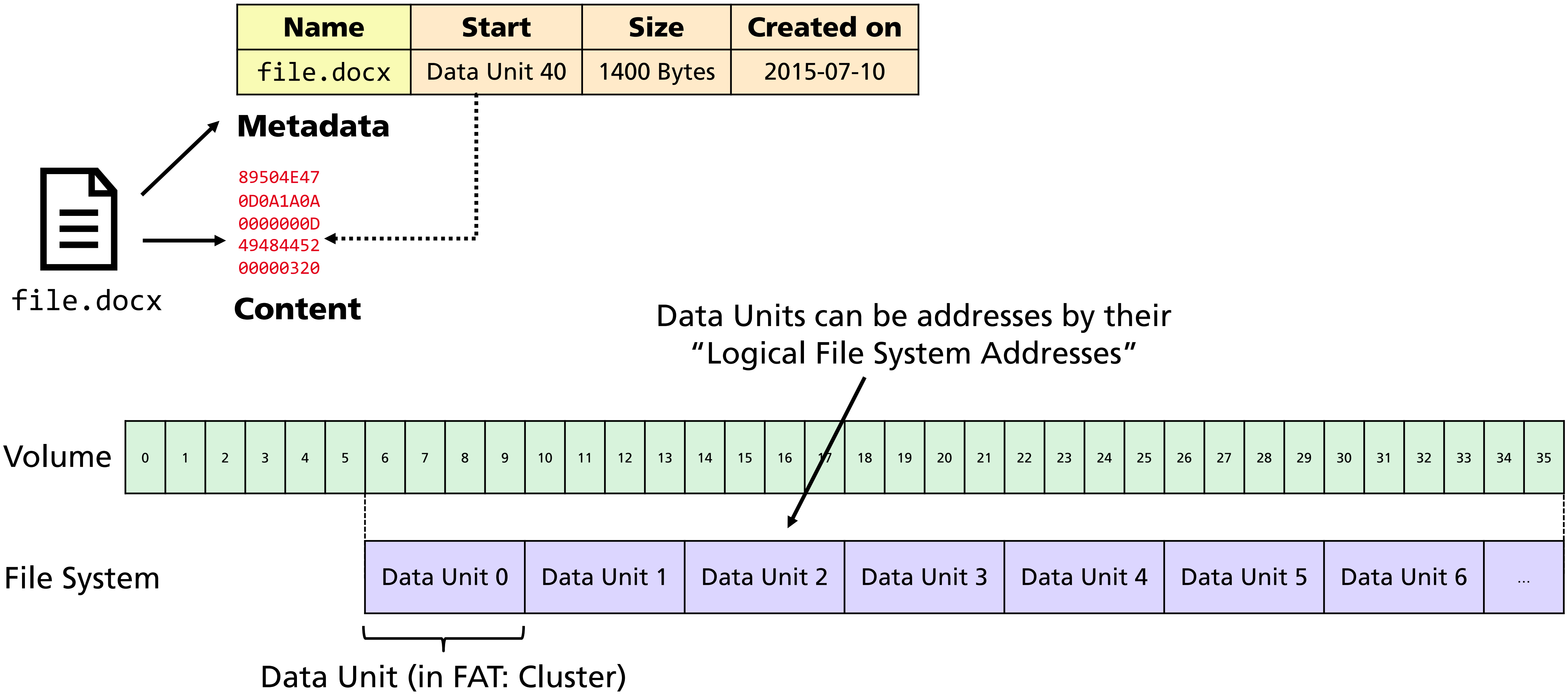
File System: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10



FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Data Units can be addresses by their “Logical File System Addresses”

Volume	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35

File System	Data Unit 0	Data Unit 1	Data Unit 2	Data Unit 3	Data Unit 4	Data Unit 5	Data Unit 6	…

Data Unit (in FAT: Cluster)

FAT file system

mkfs.fat

mkfs.fat(8) - Linux man page

Name

mkfs.fat - create an MS-DOS FAT filesystem

Synopsis

mkfs.fat [OPTIONS] DEVICE [BLOCK-COUNT]

Description

mkfs.fat is used to create a FAT filesystem on a device or in an image file.

[...]

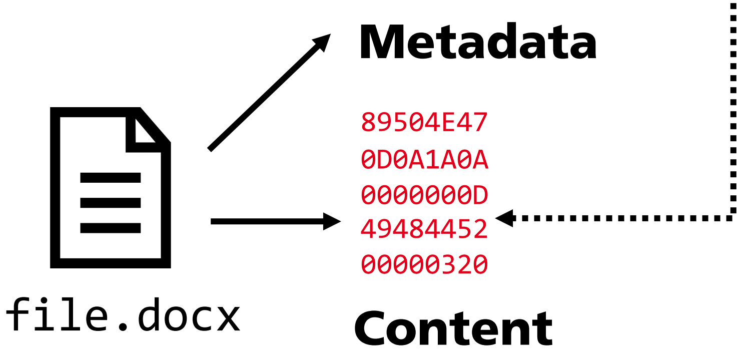
Options

-s *SECTORS-PER-CLUSTER* Specify the number of disk sectors per cluster. Must be a power of 2, i.e. 1, 2, 4, 8, ... 128.

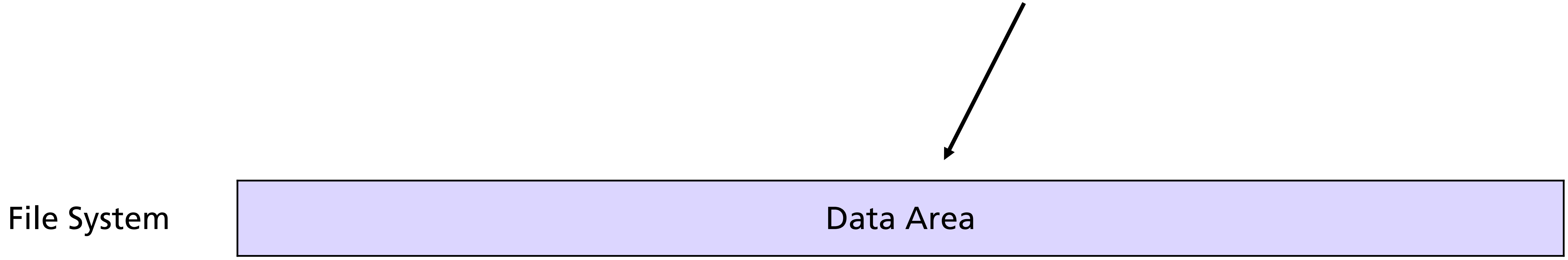
-S *LOGICAL-SECTOR-SIZE* Specify the number of bytes per logical sector. Must be a power of 2 and greater than or equal to 512, i.e. 512, 1024, 2048, 4096, 8192, 16384, or 32768. Values larger than 4096 are not conforming to the FAT file system specification and may not work everywhere.

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

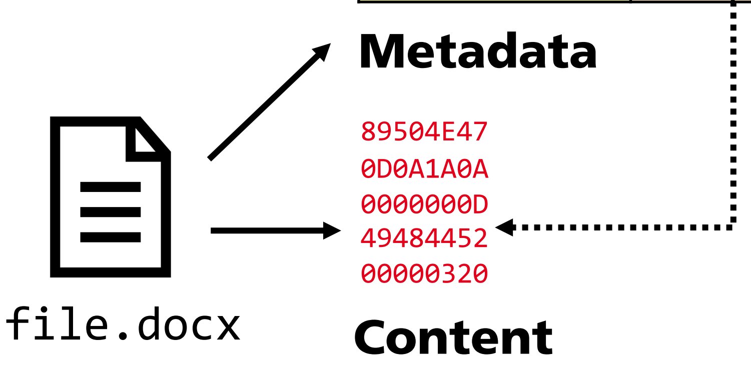


Used to store data for files and directories



FAT file system

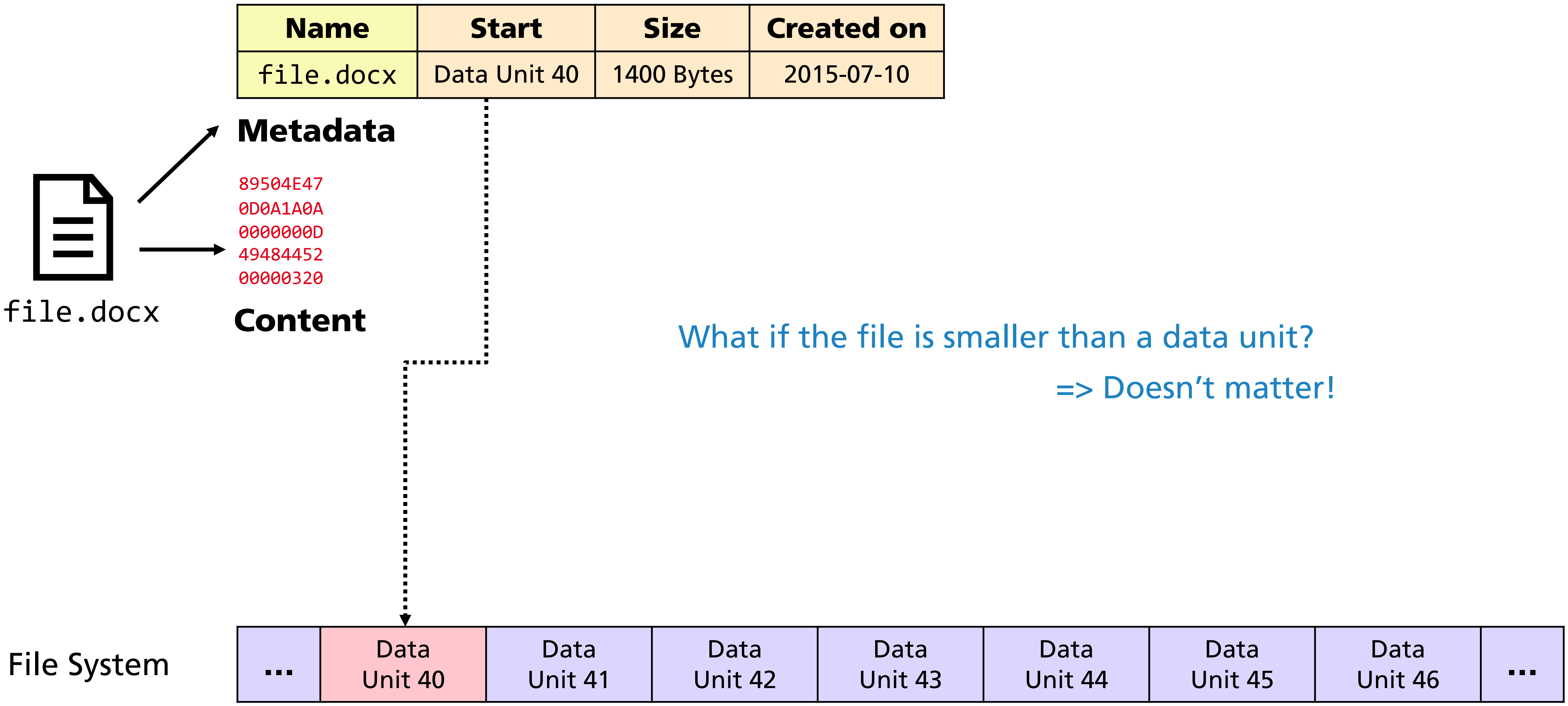
Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10



```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

File System	...	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	...

FAT file system



Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

`file.docx`

File System

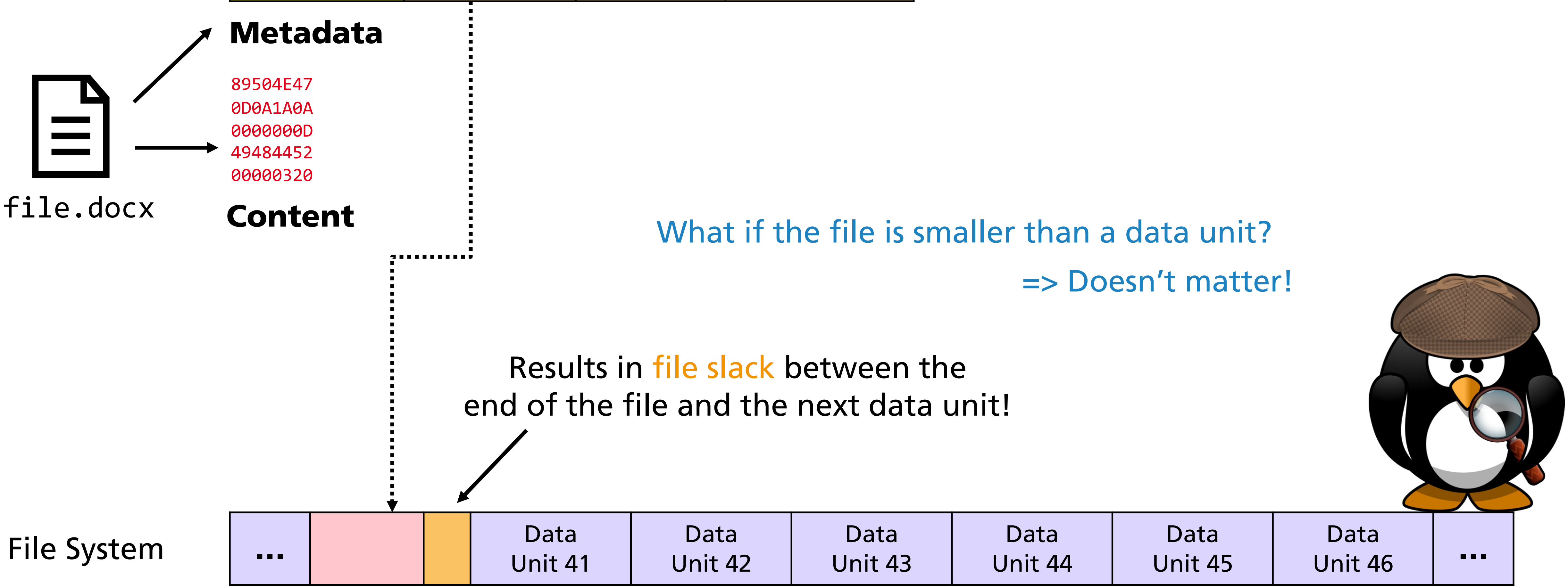
...	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	...

What if the file is smaller than a data unit?

=> Doesn’t matter!

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10



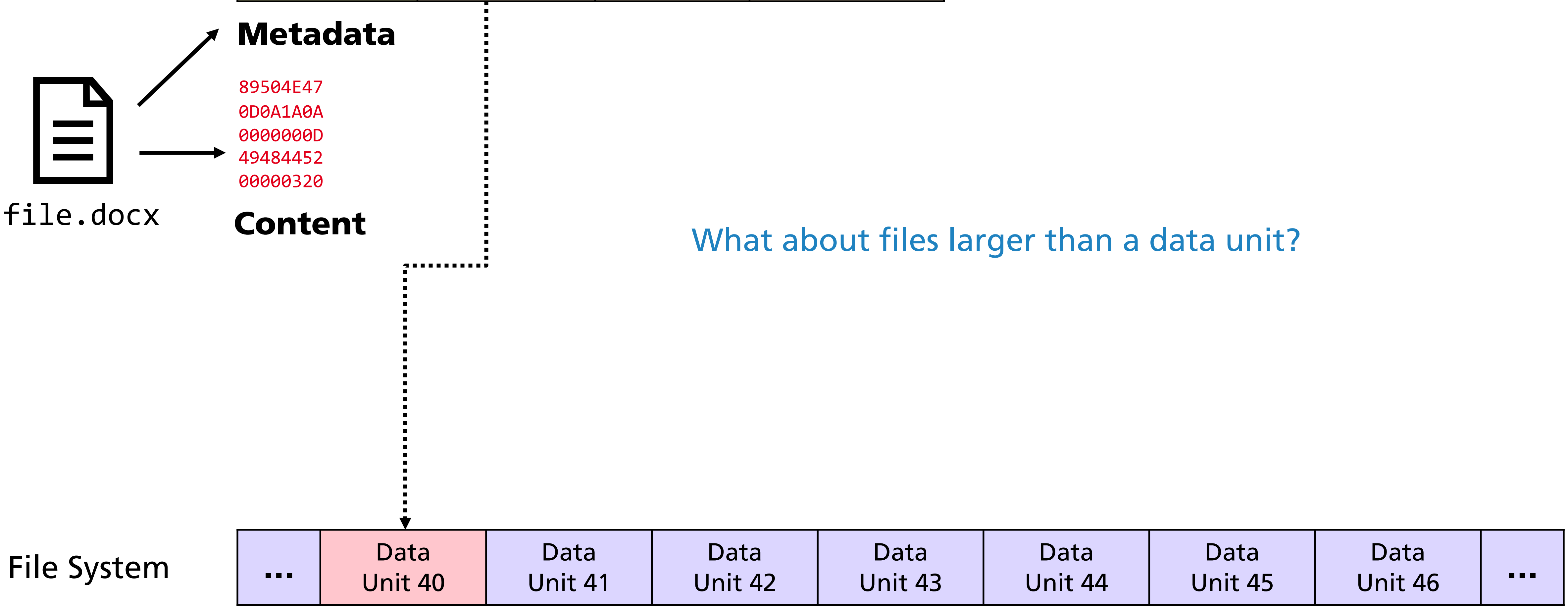
What if the file is smaller than a data unit?

=> Doesn’t matter!

Results in file slack between the end of the file and the next data unit!

FAT file system

Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10



Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

file.docx

Content

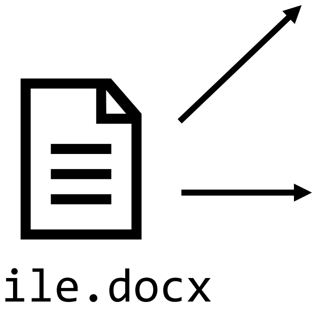
File System

...	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	...

What about files larger than a data unit?

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10



Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

The File Allocation Table is used to enable cluster chains/runs.

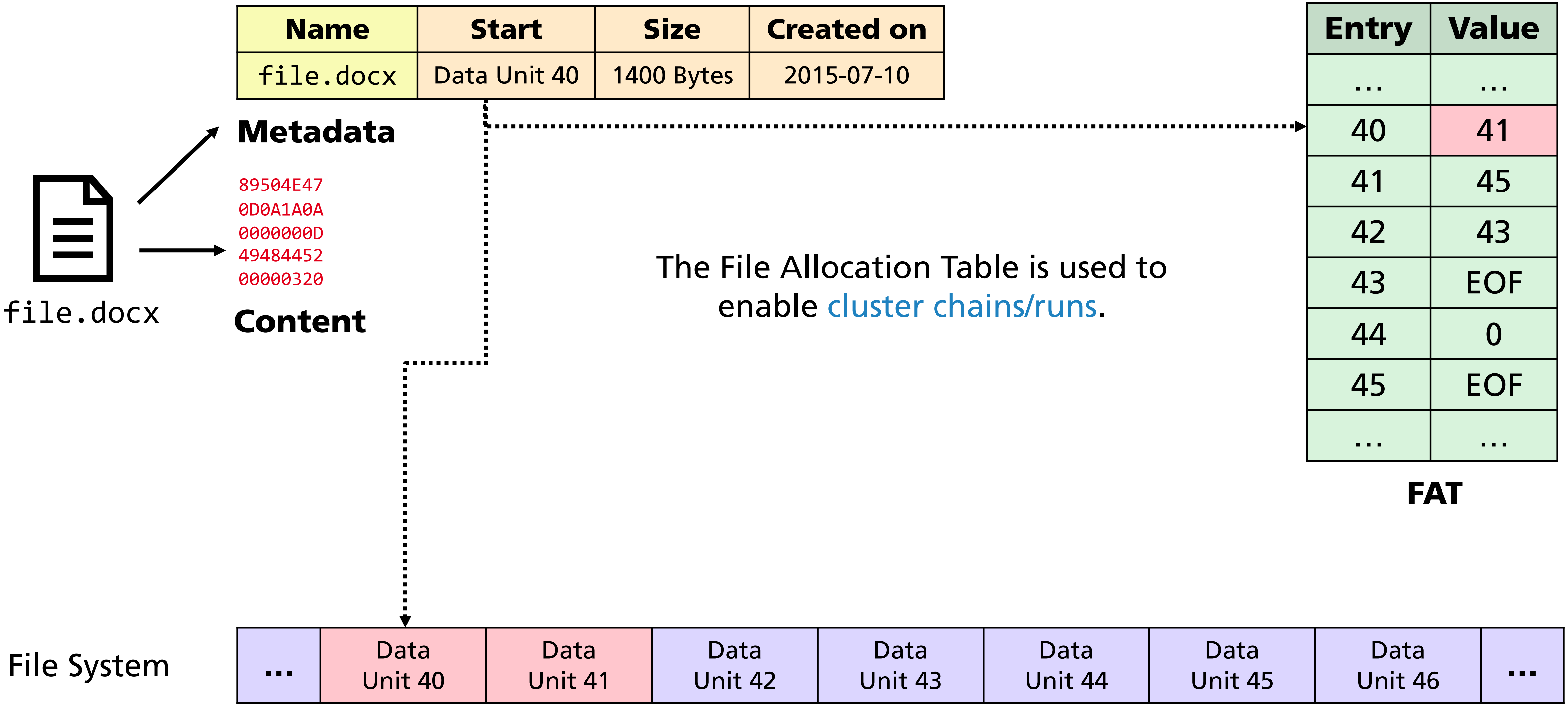
Entry	Value
…	…
40	41
41	45
42	43
43	EOF
44	0
45	EOF
…	…

FAT

File System

…	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	…

FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

file.docx

The File Allocation Table is used to enable cluster chains/runs.

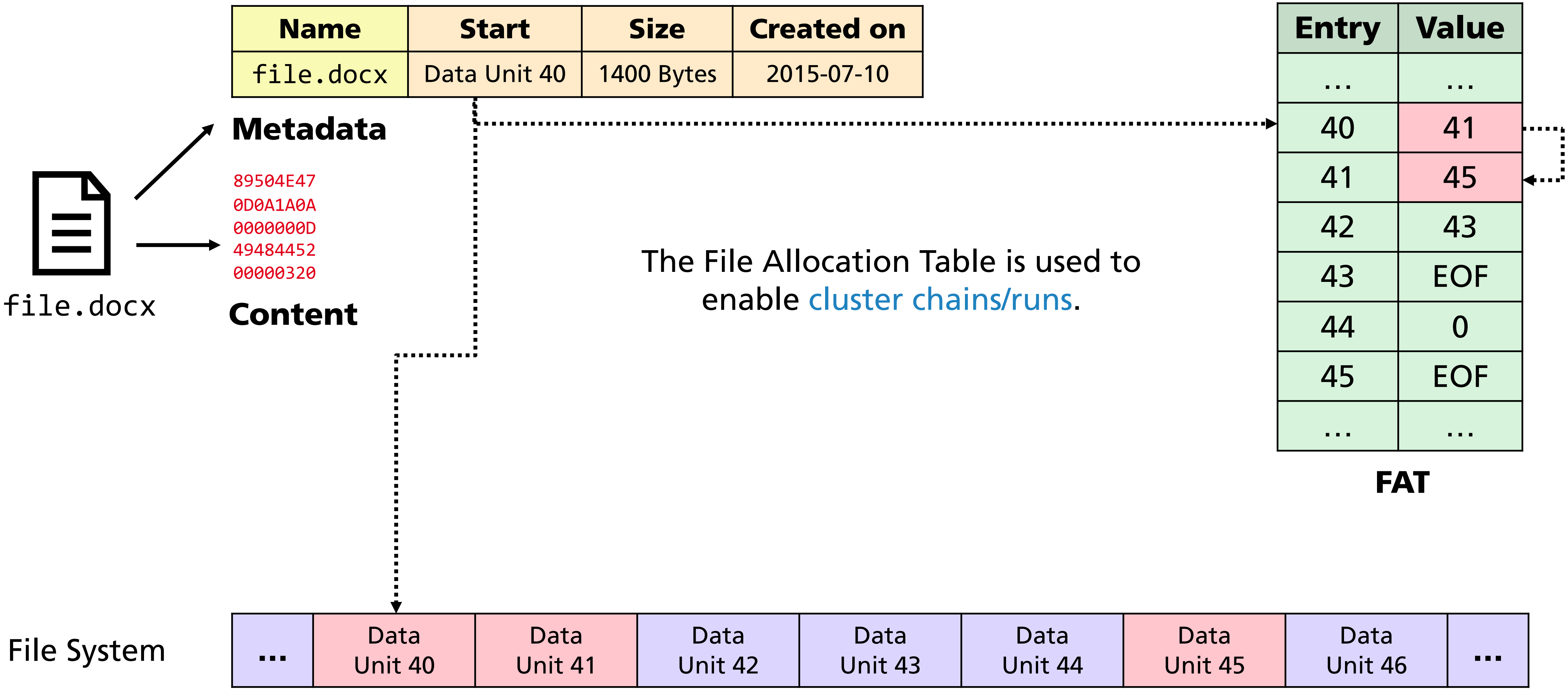
Entry	Value
…	…
40	41
41	45
42	43
43	EOF
44	0
45	EOF
…	…

FAT

File System

…	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	…

FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

The File Allocation Table is used to enable cluster chains/runs.

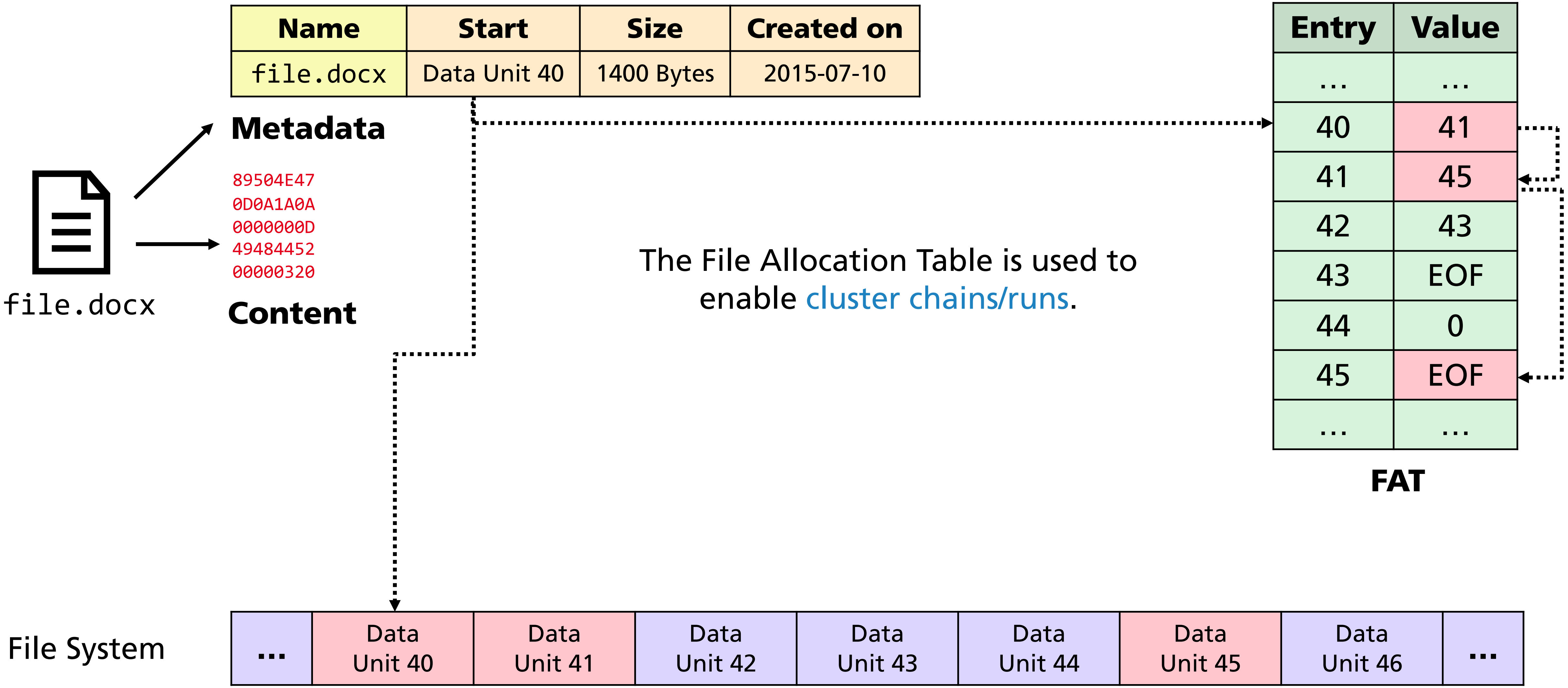
Entry	Value
…	…
40	41
41	45
42	43
43	EOF
44	0
45	EOF
…	…

FAT

File System

…	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	…

FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

The File Allocation Table is used to enable cluster chains/runs.

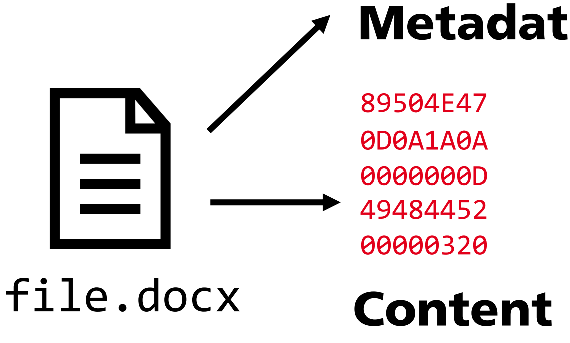
Entry	Value
…	…
40	41
41	45
42	43
43	EOF
44	0
45	EOF
…	…

FAT

File System	…	Data Unit 40	Data Unit 41	Data Unit 42	Data Unit 43	Data Unit 44	Data Unit 45	Data Unit 46	…

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

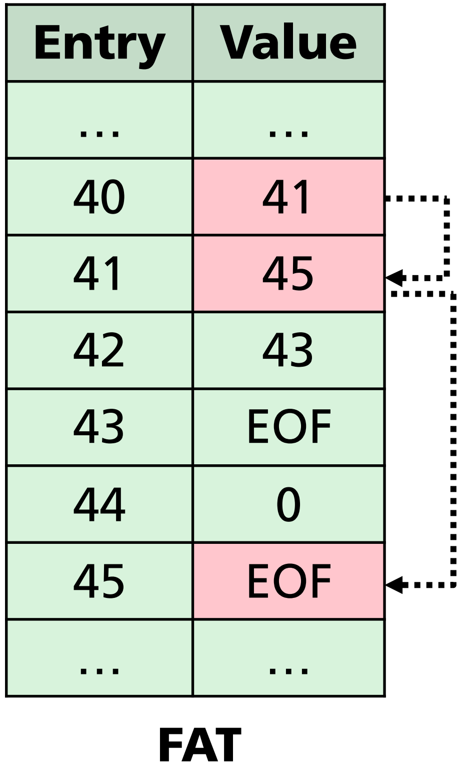


Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

The File Allocation Table is used to enable cluster chains/runs.



Entry	Value
...	...
40	41
41	45
42	43
43	EOF
44	0
45	EOF
...	...

FAT

Stores the File Allocation Table(s)

File System	FAT Area	Data Area

FAT file system

FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00																
0x10																
…																

FAT file system

FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	reserved				reserved											
0x10																
…																

FAT file system

FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF											
0x10																
…																

Some special values, we don’t care about!

FAT file system

FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	`0x0FFFFFF8`				`0x0FFFFFFF`				Cluster following cluster 2				Cluster following cluster 3			
0x10	Cluster following cluster 4				Cluster following cluster 5				Cluster following cluster 6				Cluster following cluster 7			
…																

`0x?0000000` Free cluster marker

Only 28 of the available 32 bits are used!

FAT file system

FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				Cluster following cluster 2				Cluster following cluster 3			
0x10	Cluster following cluster 4				Cluster following cluster 5				Cluster following cluster 6				Cluster following cluster 7			
…																

`0x?0000000`	Free cluster marker
`0x?FFFFFF7`	Bad cluster marker
`0x?FFFFFF8` or higher	End of file marker

Set to `0x0FFFFFFF` by Microsoft FAT drivers

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

IMG9109.jpeg
start cluster: 3

file.bin
start cluster: 2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

`IMG9109.jpeg`
start cluster: 3

`file.bin`
start cluster: 2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	`0x0FFFFFF8`				`0x0FFFFFFF`				`0x00000005`				`0x00000004`			
0x10	`0x00000007`				`0x0FFFFFF8`				`0x0000000A`				`0x00000006`			
0x20	`0x00000000`				`0x0000F541`				`0x0FFFFFF8`				`0x00000000`			

`IMG9109.jpeg`
start cluster: 3

3

`file.bin`
start cluster: 2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

IMG9109.jpeg
start cluster: 3

3 ➡ 4

file.bin
start cluster: 2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

`IMG9109.jpeg`
start cluster: 3

3 ➡ 4 ➡ 7

`file.bin`
start cluster: 2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

`IMG9109.jpeg`
start cluster: 3

3 → 4 → 7 → 6

`file.bin`
start cluster: 2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

`IMG9109.jpeg`
start cluster: 3

3 ➔ 4 ➔ 7 ➔ 6 ➔ 10

`file.bin`
start cluster: 2

FAT file system

Example: FAT (FAT32)

<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td colspan="4">0x0FFFFFF8</td><td colspan="4">0x0FFFFFFF</td><td colspan="4">0x00000005</td><td colspan="4">0x00000004</td></tr>
<tr><td>0x10</td><td colspan="4">0x00000007</td><td colspan="4">0x0FFFFFF8</td><td colspan="4">0x0000000A</td><td colspan="4">0x00000006</td></tr>
<tr><td>0x20</td><td colspan="4">0x00000000</td><td colspan="4">0x0000F541</td><td colspan="4">0x0FFFFFF8</td><td colspan="4">0x00000000</td></tr>
</table>

`IMG9109.jpeg`
start cluster: 3

3 → 4 → 7 → 6 → 10

`file.bin`
start cluster: 2

2

FAT file system

Example: FAT (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

IMG9109.jpeg
start cluster: 3

3 ➔ 4 ➔ 7 ➔ 6 ➔ 10

file.bin
start cluster: 2

2 ➔ 5

FAT file system

Analysis: Unallocated Area

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			

Unallocated clusters can be identified by checking for zeroes in the FAT!

File System

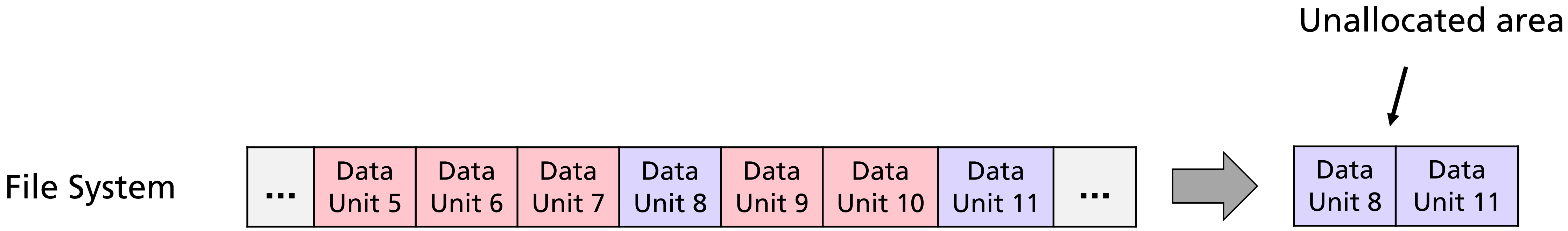
...	Data Unit 5	Data Unit 6	Data Unit 7	Data Unit 8	Data Unit 9	Data Unit 10	Data Unit 11	...

FAT file system

Analysis: Unallocated Area

- Unallocated areas may contain content of deleted files
- Can also be used to hide data since it is not actively used

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x0FFFFFF8				0x0FFFFFFF				0x00000005				0x00000004			
0x10	0x00000007				0x0FFFFFF8				0x0000000A				0x00000006			
0x20	0x00000000				0x0000F541				0x0FFFFFF8				0x00000000			



The Sleuth Kit

blkls

https://www.sleuthkit.org

blkls(1) - Linux man page

Name

blkls - List or output file system data units.

Synopsis

blkls [-aAelsvV] [-f fstype **] [-i** imgtype **] [-o** imgoffset **]** [-b dev_sector_size] image [images] [start-stop]

Description

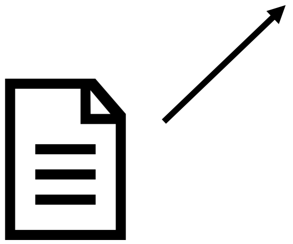
blkls opens the named image(s) and copies file system data units (blocks). By default, **blkls** copies the contents of unallocated data blocks. **blkls** was called **dls** in TSK versions prior to 3.0.0. **blkls** was called **unrm** in TCT.

-e Copy every block, including file system metadata blocks. The output is the entire file system.
-a Display all allocated blocks (same as -e if -A is also given).
-A Display all unallocated blocks (same as -e if -a is also given). This is the default behavior.
[...]

FAT file system

Name	Start	Size	Created on
`file.docx`	Data Unit 40	1400 Bytes	2015-07-10

Metadata



`file.docx`

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	short name											attr.	res.	created time		
0x10	created date		date last accessed		first cluster (high bytes)		time of last write		date of last write		first cluster (low bytes)		file size			

Directory Entry

The Sleuth Kit

istat

https://www.sleuthkit.org

istat(1) - Linux man page

Name

istat - Display details of a meta-data structure (i.e. inode)

Synopsis

istat [-B *num* **] [-f** *fstype* **] [-i imgtype] [-o imgoffset] [-b dev_sector_size] [-vV] [-z** *zone* **] [-s** *seconds* **]** *image [images] inode*

Description

istat displays the uid, gid, mode, size, link number, modified, accessed, changed times, and all the disk units a structure has allocated.

The Sleuth Kit

istat

```
$ istat disk.dd 4

Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 1096213
Name: IMAGES~1.JPG


Directory Entry Times:
Written:        2020-11-15 18:05:06 (CET)
Accessed:       2020-11-15 00:00:00 (CET)
Created:        2020-11-15 18:05:06 (CET)

Sectors:
440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455
456 457 458 459 460 461 462 463
464 465 466 467 468 469 470 471
[...]
```

https://www.sleuthkit.org

The Sleuth Kit

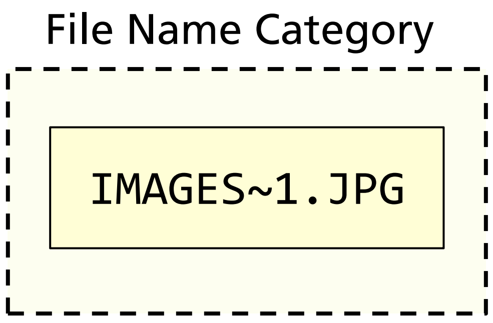
istat

```
$ istat disk.dd 4

Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 1096213
Name: IMAGES~1.JPG


Directory Entry Times:
Written:        2020-11-15 18:05:06 (CET)
Accessed:       2020-11-15 00:00:00 (CET)
Created:        2020-11-15 18:05:06 (CET)

Sectors:
440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455
456 457 458 459 460 461 462 463
464 465 466 467 468 469 470 471
[...]
```



https://www.sleuthkit.org

The Sleuth Kit

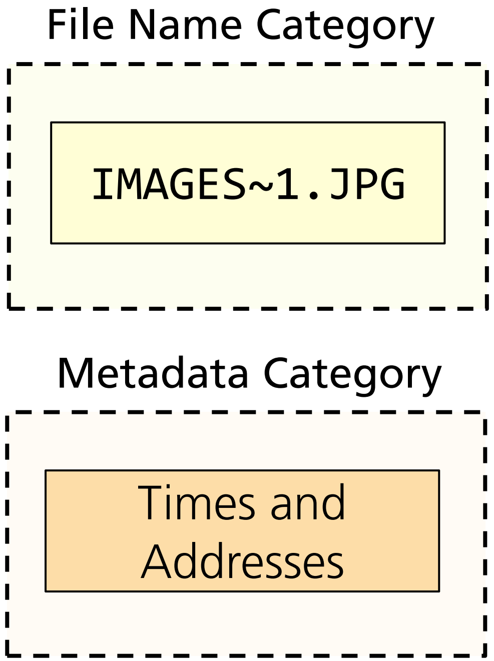
istat

```
$ istat disk.dd 4

Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 1096213
Name: IMAGES~1.JPG


Directory Entry Times:
Written:        2020-11-15 18:05:06 (CET)
Accessed:       2020-11-15 00:00:00 (CET)
Created:        2020-11-15 18:05:06 (CET)

Sectors:
440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455
456 457 458 459 460 461 462 463
464 465 466 467 468 469 470 471
[...]
```



File Name Category

IMAGES~1.JPG

Metadata Category

Times and Addresses

https://www.sleuthkit.org

The Sleuth Kit

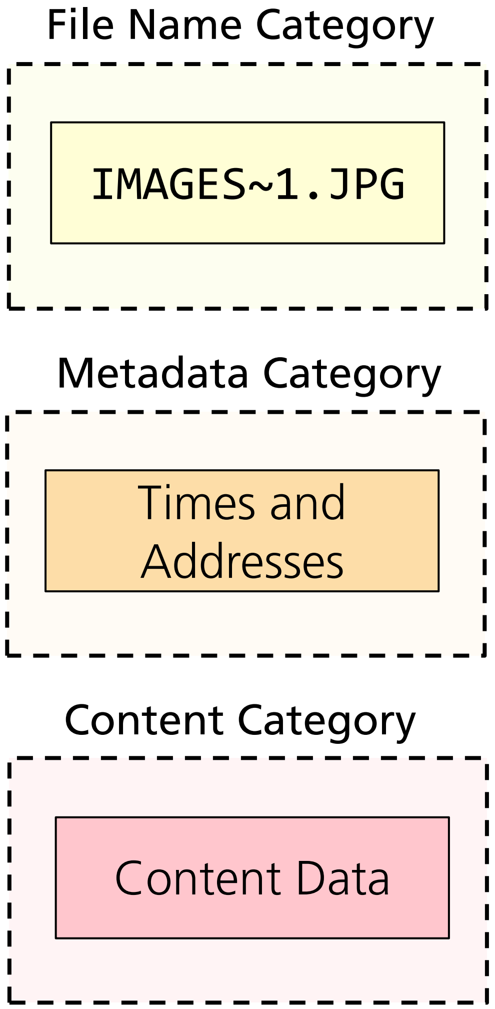
istat

```
$ istat disk.dd 4

Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 1096213
Name: IMAGES~1.JPG


Directory Entry Times:
Written:        2020-11-15 18:05:06 (CET)
Accessed:       2020-11-15 00:00:00 (CET)
Created:        2020-11-15 18:05:06 (CET)

Sectors:
440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455
456 457 458 459 460 461 462 463
464 465 466 467 468 469 470 471
[...]
```



File Name Category

IMAGES~1.JPG

Metadata Category

Times and Addresses

Content Category

Content Data

https://www.sleuthkit.org

The Sleuth Kit

icat

icat(1) - Linux man page

Name

icat - Output the contents of a file based on its inode number.

Synopsis

icat [-hrsvV] [-f *fstype* **] [-i** *imgtype* **] [-o** *imgoffset* **] [-b dev_sector_size]** *image* [images] inode

Description

icat opens the named *image(s)* and copies the file with the specified *inode* number to standard output.

https://www.sleuthkit.org

The Sleuth Kit

istat

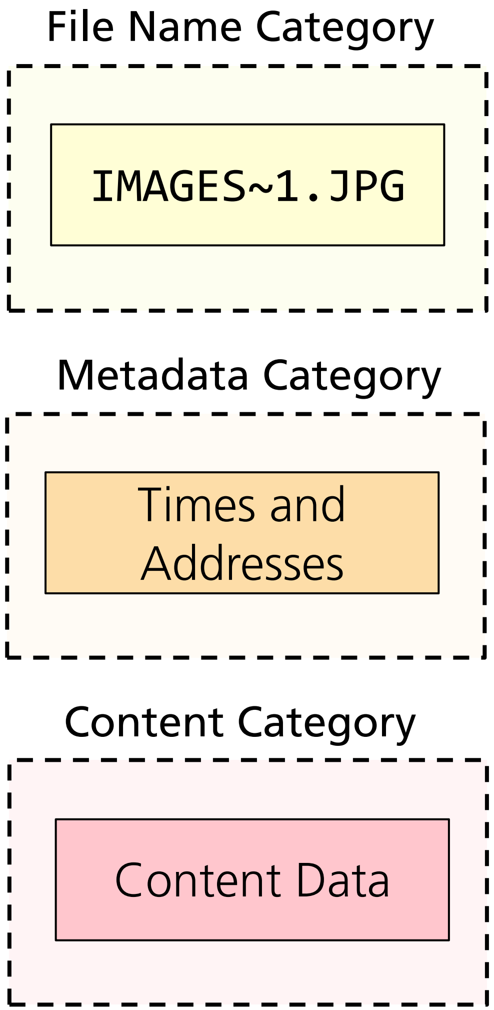
How are directory entries numbered?

```
$ istat disk.dd 4

Directory Entry: 4
Allocated
File Attributes: File, Archive
Size: 1096213
Name: IMAGES~1.JPG


Directory Entry Times:
Written:        2020-11-15 18:05:06 (CET)
Accessed:       2020-11-15 00:00:00 (CET)
Created:        2020-11-15 18:05:06 (CET)

Sectors:
440 441 442 443 444 445 446 447
448 449 450 451 452 453 454 455
456 457 458 459 460 461 462 463
464 465 466 467 468 469 470 471
[...]
```



File Name Category

IMAGES~1.JPG

Metadata Category

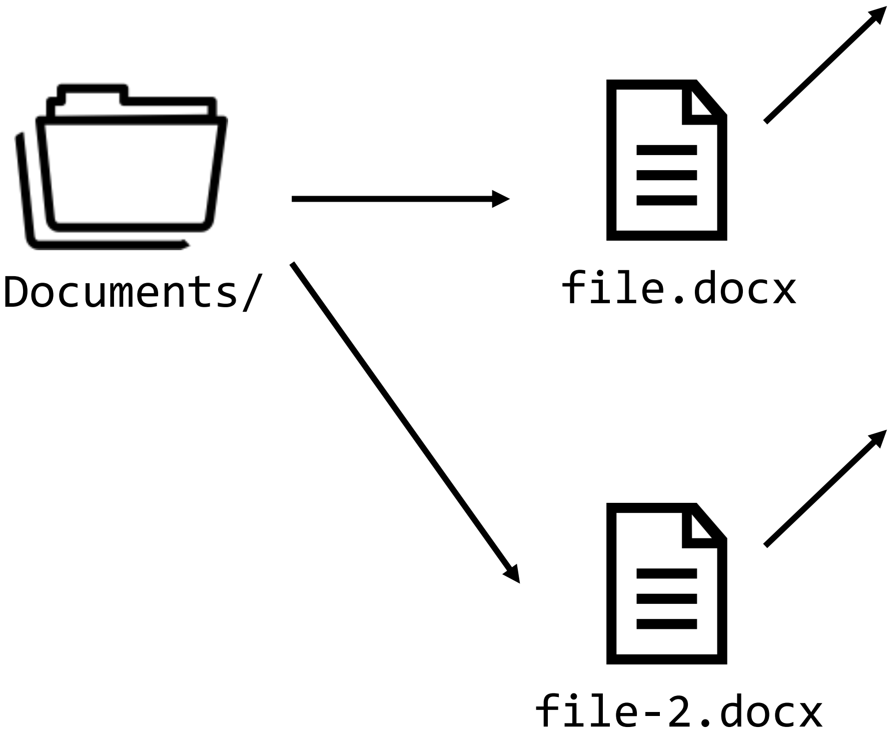
Times and Addresses

Content Category

Content Data

https://www.sleuthkit.org

FAT file system



Name	Start	Size	Created on
file.docx	Data Unit 40	1400 Bytes	2015-07-10

Metadata

Name	Start	Size	Created on
file-2.docx	Data Unit 60	1400 Bytes	2015-07-11

Metadata

FAT file system

Name	Start	Size
Documents	Data Unit 20	0 Bytes

Metadata

Documents/

Directories have the corresponding attribute flag set

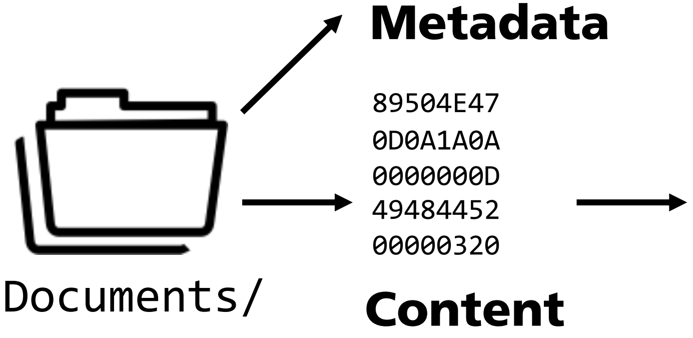
<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td colspan="11">short name</td><td>attr.</td><td>res.</td><td colspan="3">created time</td></tr>
<tr><td>0x10</td><td colspan="2">created date</td><td colspan="2">date last accessed</td><td colspan="2">first cluster (high bytes)</td><td colspan="2">time of last write</td><td colspan="2">date of last write</td><td colspan="2">first cluster (low bytes)</td><td colspan="4">0x00000000</td></tr>
</table>

Directory Entry

Should be set to zero

FAT file system

Name	Start	Size
Documents	Data Unit 20	0 Bytes



Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

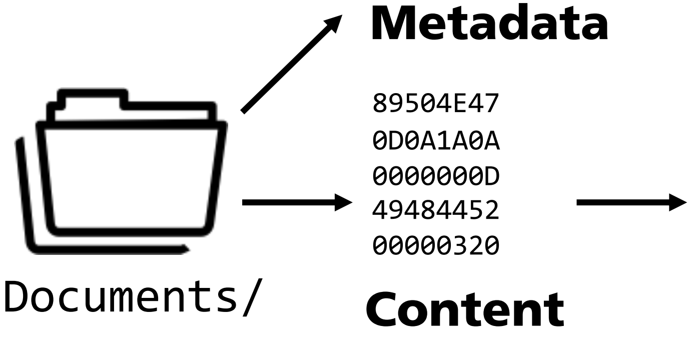
Documents/

Name	Start	Size
.	Data Unit 20	0
..	Data Unit 10	0
IMG10.png	Data Unit 1	2 MB
IMG11.png	Data Unit 3	500 kB
IMG12.png	Data Unit 4	500 KB
IMG13.png	Data Unit 6	1 MB
file.docx	Data Unit 40	1400 Bytes
…	…	…

The content of a directory are directory entries

FAT file system

Name	Start	Size
Documents	Data Unit 20	0 Bytes



```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Name	Start	Size
.	Data Unit 20	0
..	Data Unit 10	0
IMG10.png	Data Unit 1	2 MB
IMG11.png	Data Unit 3	500 kB
IMG12.png	Data Unit 4	500 KB
IMG13.png	Data Unit 6	1 MB
file.docx	Data Unit 40	1400 Bytes
…	…	…

It contains directory entries for itself and its parent directory

FAT file system

- When a new directory is created, the assigned cluster is wiped with zeroes.
- The creation timestamp of the . and .. entries should match the creation timestamp within the parent directory

Documents/

Name	Start	Size
Documents	Data Unit 20	0 Bytes

Metadata

```
89504E47
0D0A1A0A
0000000D
49484452
00000320
```

Content

Name	Start	Size
.	Data Unit 20	0
..	Data Unit 10	0
IMG10.png	Data Unit 1	2 MB
IMG11.png	Data Unit 3	500 kB
IMG12.png	Data Unit 4	500 KB
IMG13.png	Data Unit 6	1 MB
file.docx	Data Unit 40	1400 Bytes
Secret	Data Unit 90	0
…	…	…

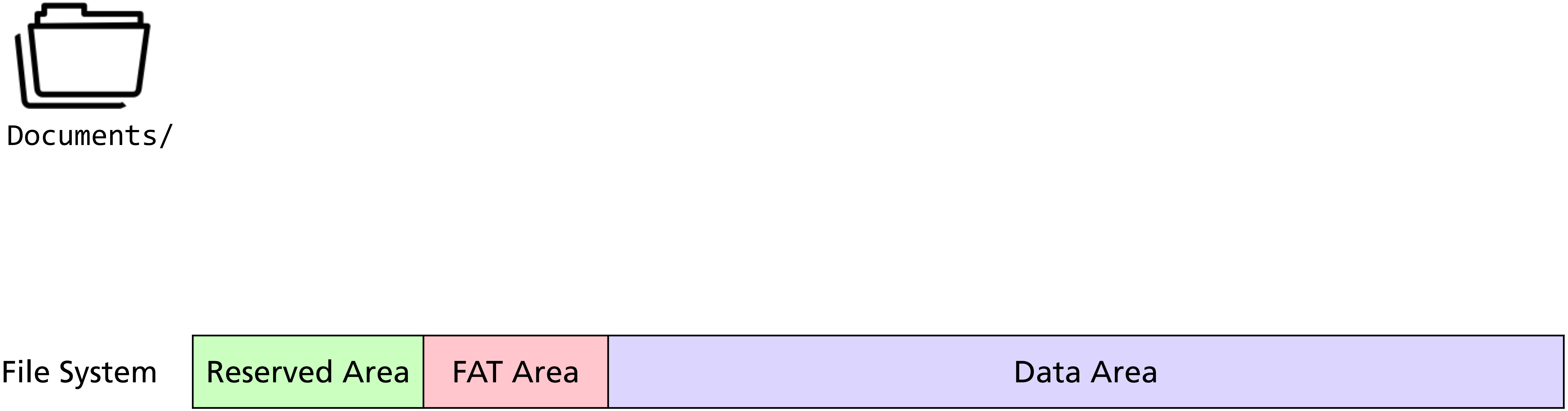
Cluster 20

Cluster 90

Name	Start	Size
.	Cluster 90	0
..	Cluster 20	0
top.pdf	Cluster 190	1 MB
…	…	…

FAT file system

Metadata Addresses



Documents/

File System

Reserved Area

FAT Area

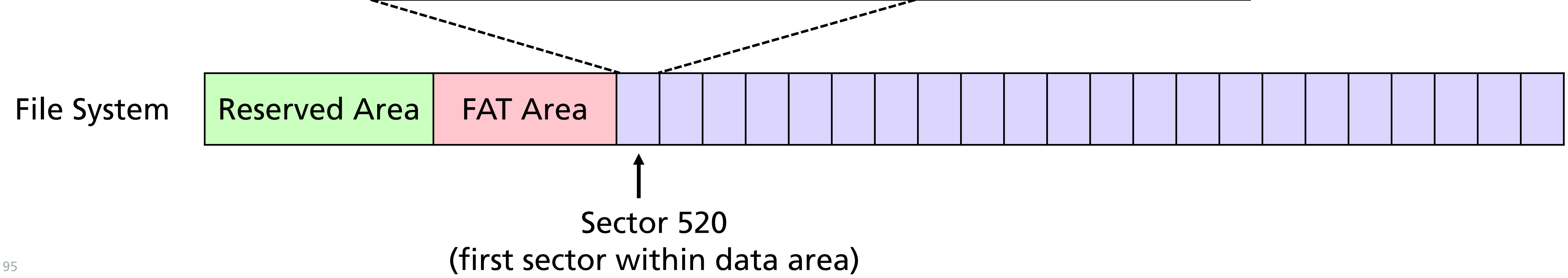
Data Area

FAT file system

Metadata Addresses

Documents/

Name	Start	Size	Metadata Address
IMG10.png	Cluster 40	1400 Bytes	3
IMG11.png			4
IMG12.png			5
IMG13.png			6
IMG14.png			7
...	...	...	...

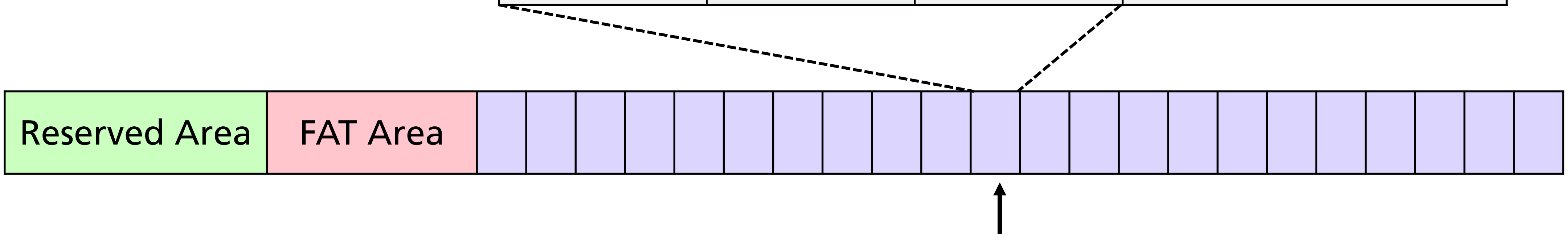


FAT file system

Metadata Addresses

Documents/

Name	Start	Size	Metadata Address
			163
			164
			165
			166
			167
…	…	…	…

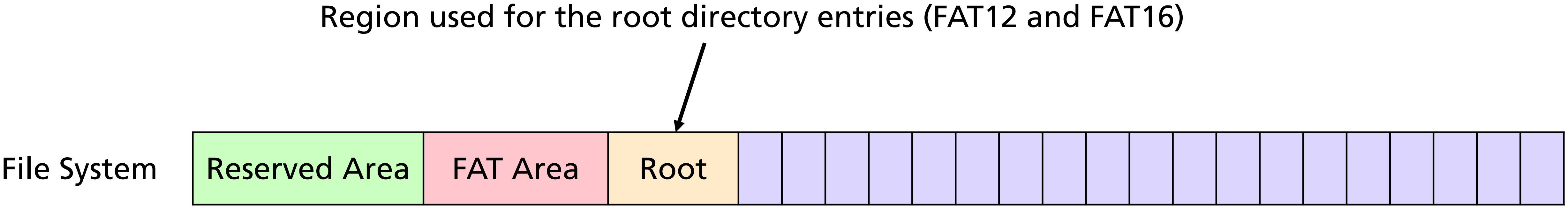


Sector 530

FAT file system

Metadata Addresses

Documents/


Region used for the root directory entries (FAT12 and FAT16)
File System
Reserved Area
FAT Area
Root

The Sleuth Kit

fls

fls(1) - Linux man page

Name

fls - List file and directory names in a disk image.

Synopsis

fls [-adDFlpruvV] [-m *mnt* **] [-z** *zone* **] [-f** *fstype* **] [-s** *seconds* **] [-i** *imgtype* **]**
[-o *imgoffset* **] [-b dev_sector_size]** *image* [images] [*inode*]

Description

fls lists the files and directory names in the *image* and can display file names of recently deleted files for the directory using the given *inode*. If the inode argument is not given, the inode value for the root directory is used. For example, on an NTFS file system it would be 5 and on a Ext3 file system it would be 2.

https://www.sleuthkit.org

The Sleuth Kit

fls

```
$ fls -r disk.dd

r/r 4:  suspicios.jpg
d/d 6:  videos
+ r/r 43014:    vid1.mp4
+ r/r 43016:    vid2.mp4
r/r * 7:        _UN-56~1.JPG
v/v 3270339:    $MBR
v/v 3270340:    $FAT1
v/v 3270341:    $FAT2
V/V 3270342:    $OrphanFiles
```

https://www.sleuthkit.org

The Sleuth Kit

fls

```
$ fls -r disk.dd

r/r 4:  suspicios.jpg
d/d 6:  videos
+ r/r 43014:    vid1.mp4
+ r/r 43016:    vid2.mp4
r/r * 7:        _UN-56~1.JPG
v/v 3270339:    $MBR
v/v 3270340:    $FAT1
v/v 3270341:    $FAT2
V/V 3270342:    $OrphanFiles
```

Metadata (inode) addresses

https://www.sleuthkit.org

The Sleuth Kit

fls

```
$ fls -r disk.dd

r/r 4:  suspicios.jpg
d/d 6:  videos
+ r/r 43014:    vid1.mp4
+ r/r 43016:    vid2.mp4
r/r * 7:        _UN-56~1.JPG
v/v 3270339:    $MBR
v/v 3270340:    $FAT1
v/v 3270341:    $FAT2
V/V 3270342:    $OrphanFiles
```

Deleted files are indicated by an asterisk

https://www.sleuthkit.org

FAT file system

Scenario: File Deletion

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
NYC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
videos	Cluster 10	0
documents	Cluster 11	0
...	...	...

Cluster 140

	...
58	0
59	0
60	62
61	EOF
62	63
63	EOF
	...

FAT

FAT file system

Scenario: File Deletion

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
videos	Cluster 10	0
documents	Cluster 11	0
...	...	...

Cluster 140

	...
58	0
59	0
60	62
61	EOF
62	63
63	EOF
	...

FAT

- After deletion, the metadata entry is marked as unallocated by setting the first byte to `0xe5`

FAT file system

Scenario: File Deletion

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
videos	Cluster 10	0
documents	Cluster 11	0
…	…	…

Cluster 140

	…
58	0
59	0
60	0
61	EOF
62	0
63	0
	…

FAT

- After deletion, the metadata entry is marked as unallocated by setting the first byte to `0xe5`
- The corresponding clusters are marked as unallocated by setting their entry in the FAT to 0

„File deletion is OS-specific and may differ between implementations. Most implementations of FAT file systems will delete a file by setting the directory entry to an unallocated state (which overwrites the first letter of the name) and sets the table entries for the file's clusters to 0. In general, the starting cluster and the size of the file are not wiped from the directory entry. Again, this is OS-specific and an OS may choose to wipe those fields when a file is deleted.“

http://www.sleuthkit.org/informer/sleuthkit-informer-14.txt

FAT file system

Analysis: Metadata-based File Recovery

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
videos	Cluster 10	0
documents	Cluster 11	0
…	…	…

Cluster 140

	…
58	0
59	0
60	0
61	EOF
62	0
63	0
	…

FAT

- After deletion, the metadata entry is marked as unallocated by setting the first byte to `0xe5`
- The corresponding clusters are marked as unallocated by setting their entry in the FAT to 0

○ Unallocated metadata entries can still be found, e.g. by searching for a certain signature (such as `0xe5`) or by simply iterating over a data unit

○ Referenced data units may have already been overwritten without you noticing

○ Even worse: If metadata entries don't store all content addresses, it may be impossible to recover a file

FAT file system

Analysis: Metadata-based File Recovery

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
videos	Cluster 10	0
documents	Cluster 11	0
LA.png	Cluster 60	6000 Bytes
…	…	…

Cluster 140

	…
58	0
59	0
60	140
61	EOF
62	0
63	0
	…

FAT

- After deletion, the metadata entry is marked as unallocated by setting the first byte to `0xe5`
- The corresponding clusters are marked as unallocated by setting their entry in the FAT to 0

○ Unallocated metadata entries can still be found, e.g. by searching for a certain signature (such as `0xe5`) or by simply iterating over a data unit

○ Referenced data units may have already been overwritten without you noticing

○ Even worse: If metadata entries don't store all content addresses, it may be impossible to recover a file

The Sleuth Kit

fls & icat

```
$ fls -r disk.dd

r/r 4:  suspicios.jpg
d/d 6:  videos
+ r/r 43014:    vid1.mp4
+ r/r 43016:    vid2.mp4
r/r * 7:        _UN-56~1.JPG
v/v 3270339:    $MBR
v/v 3270340:    $FAT1
v/v 3270341:    $FAT2
V/V 3270342:    $OrphanFiles
```

Deleted files are indicated by an asterisk

```
$ icat disk.dd 7 > recovered.JPG
```

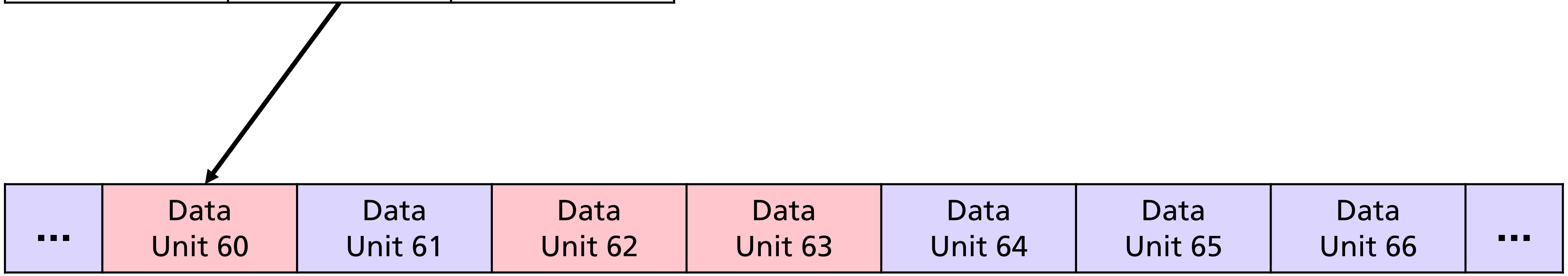
https://www.sleuthkit.org

The Sleuth Kit

FAT File Recovery

https://www.sleuthkit.org

Name	Start	Size
NYC.png	Cluster 60	5000 Bytes



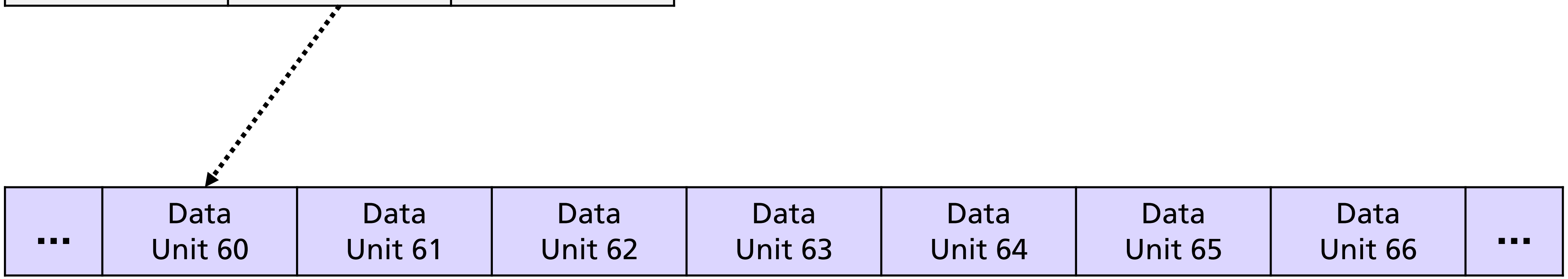
...	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	...

The Sleuth Kit

FAT File Recovery

https://www.sleuthkit.org

Name	Start	Size
0xe5YC.png	Cluster 60	5000 Bytes



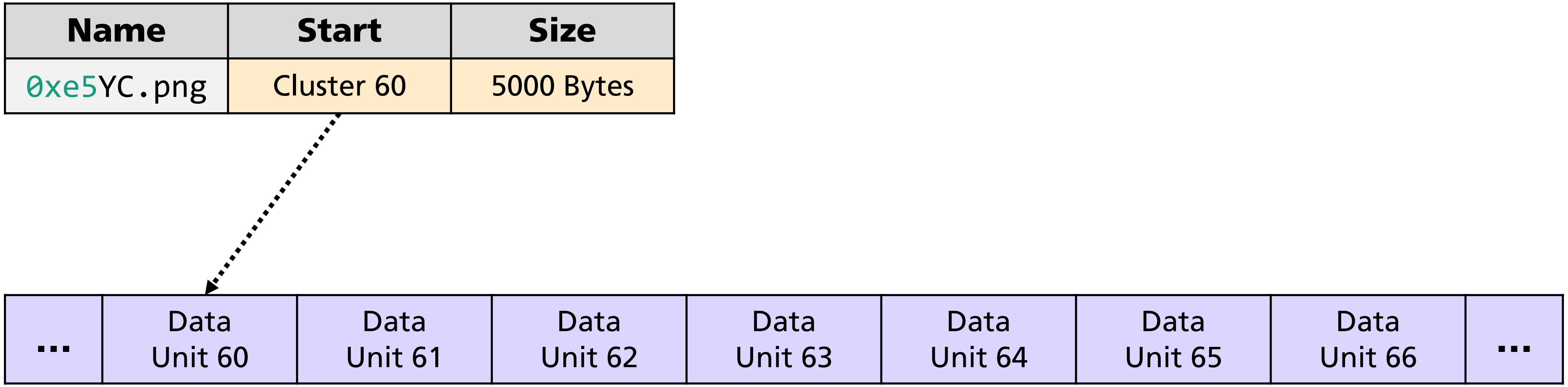
...	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	...

The Sleuth Kit

FAT File Recovery

- For FAT file recovery TSK uses the start and size fields of unallocated directory entries
- If the starting cluster is allocated, it will not recover any data

https://www.sleuthkit.org



Name	Start	Size
0xe5YC.png	Cluster 60	5000 Bytes

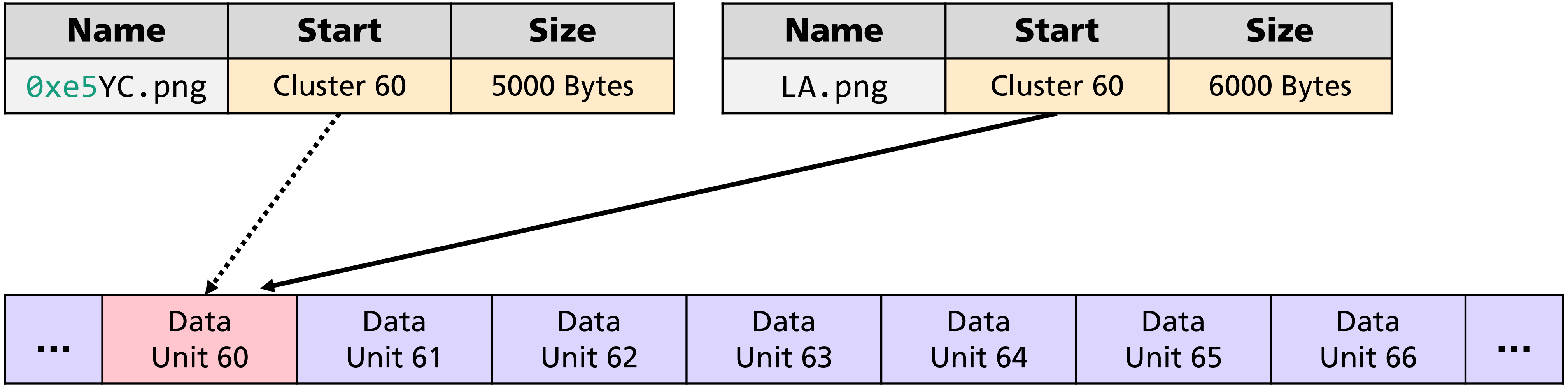
…	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	…

The Sleuth Kit

FAT File Recovery

https://www.sleuthkit.org

- For FAT file recovery TSK uses the start and size fields of unallocated directory entries
- If the starting cluster is allocated, it will not recover any data



Name	Start	Size
0xe5YC.png	Cluster 60	5000 Bytes

Name	Start	Size
LA.png	Cluster 60	6000 Bytes

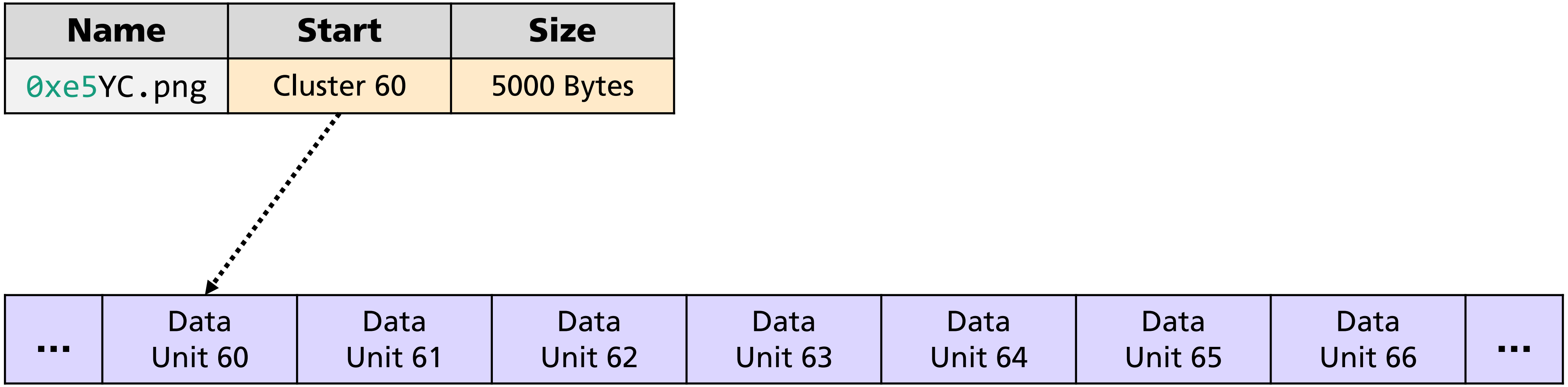
…	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	…

The Sleuth Kit

FAT File Recovery

- For FAT file recovery TSK uses the start and size fields of unallocated directory entries
- If the starting cluster is allocated, it will not recover any data
- If the starting cluster is unallocated, it will extract consecutive and unallocated clusters until the final file size is reached

https://www.sleuthkit.org



Name	Start	Size
0xe5YC.png	Cluster 60	5000 Bytes

...	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	...

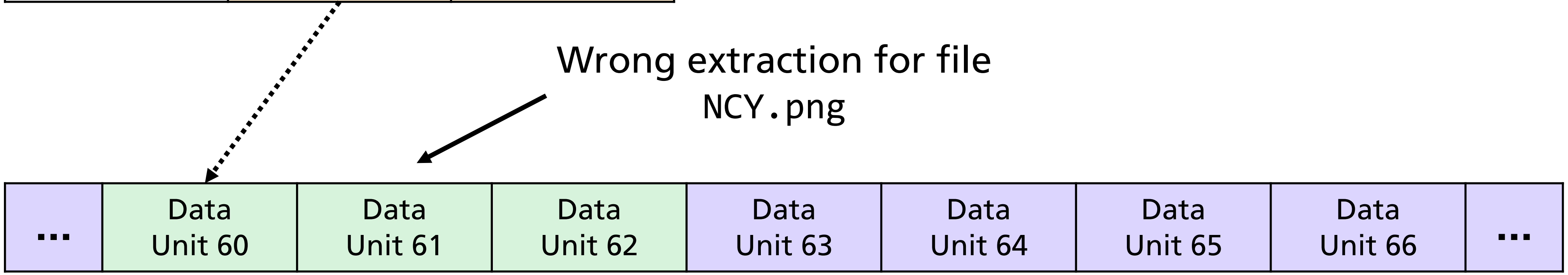
The Sleuth Kit

FAT File Recovery

- For FAT file recovery TSK uses the start and size fields of unallocated directory entries
- If the starting cluster is allocated, it will not recover any data
- If the starting cluster is unallocated, it will extract consecutive and unallocated clusters until the final file size is reached

https://www.sleuthkit.org

Name	Start	Size
0xe5YC.png	Cluster 60	5000 Bytes



Wrong extraction for file NCY.png

...	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	...

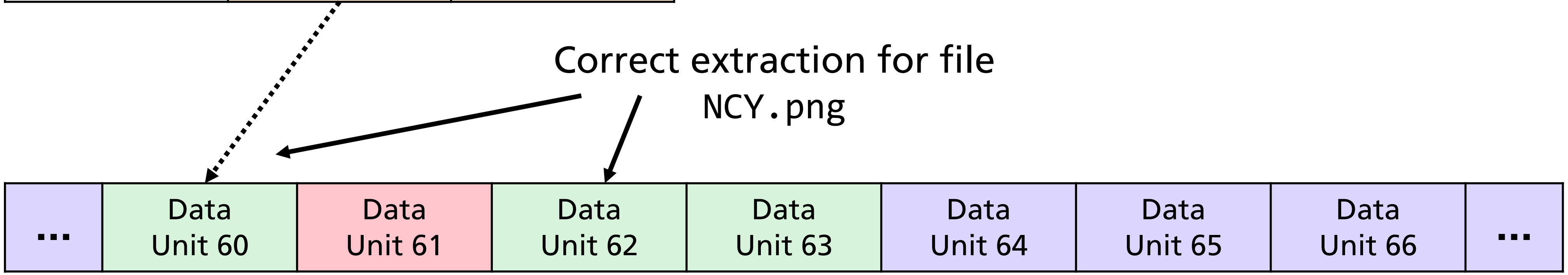
The Sleuth Kit

FAT File Recovery

https://www.sleuthkit.org

- For FAT file recovery TSK uses the start and size fields of unallocated directory entries
- If the starting cluster is allocated, it will not recover any data
- If the starting cluster is unallocated, it will extract consecutive and unallocated clusters until the final file size is reached

Name	Start	Size
0xe5YC.png	Cluster 60	5000 Bytes

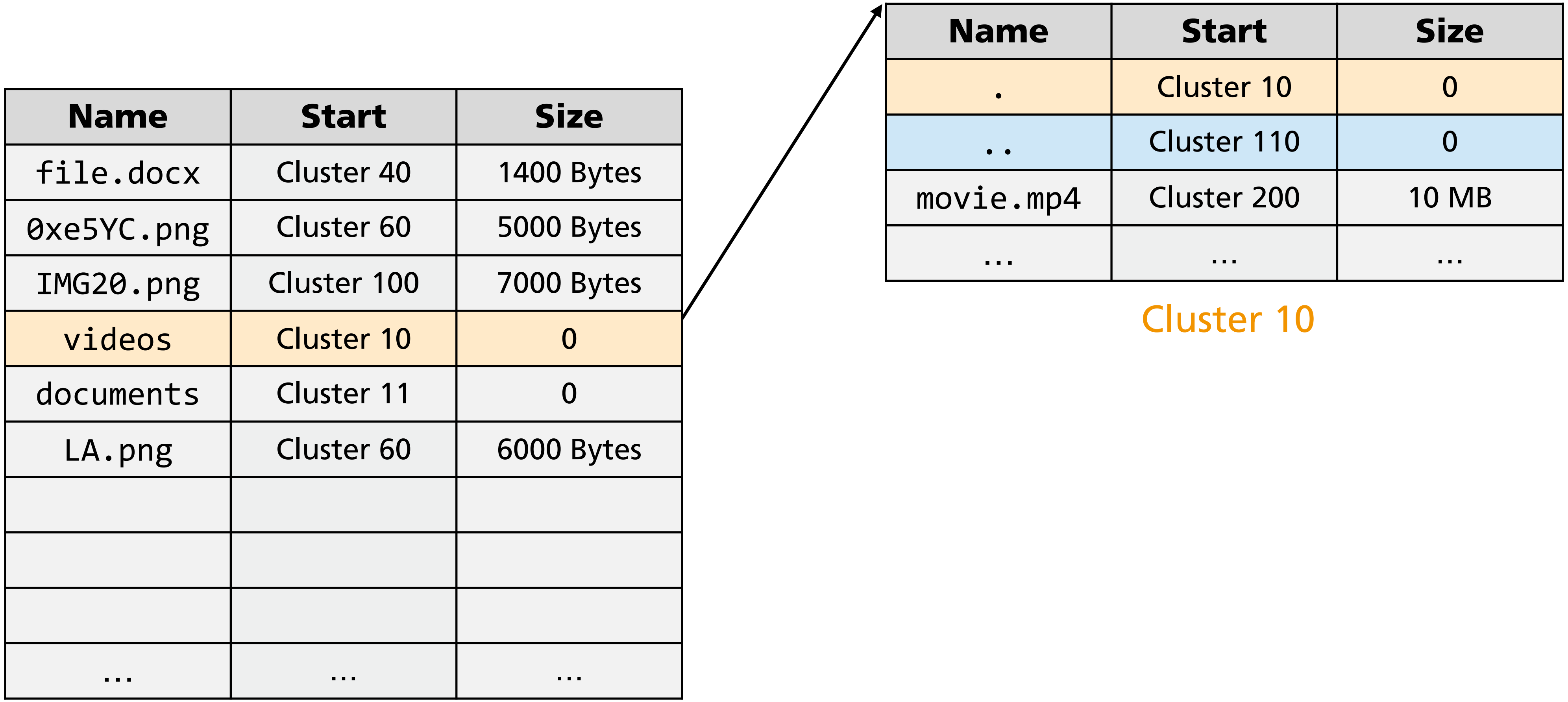


Correct extraction for file NCY.png

...	Data Unit 60	Data Unit 61	Data Unit 62	Data Unit 63	Data Unit 64	Data Unit 65	Data Unit 66	...

FAT file system

Scenario: Directory Deletion



Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
videos	Cluster 10	0
documents	Cluster 11	0
LA.png	Cluster 60	6000 Bytes
…	…	…

Cluster 140

Name	Start	Size
.	Cluster 10	0
..	Cluster 110	0
movie.mp4	Cluster 200	10 MB
…	…	…

Cluster 10

FAT file system

Scenario: Directory Deletion

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
0xe5ideos	Cluster 10	0
documents	Cluster 11	0
LA.png	Cluster 60	6000 Bytes
…	…	…

Cluster 140

Name	Start	Size
.	Cluster 10	0
..	Cluster 110	0
0xe5ovie.mp4	Cluster 200	10 MB
…	…	…

Cluster 10

- ○ Directory entries of deleted directories are also unallocated except for the first two!
- ○ Clusters belonging to directories can easily be identified by the . and .. directory entries at the beginning

FAT file system

Scenario: Directory Deletion

Name	Start	Size
file.docx	Cluster 40	1400 Bytes
0xe5YC.png	Cluster 60	5000 Bytes
IMG20.png	Cluster 100	7000 Bytes
music	Cluster 200	0
documents	Cluster 11	0
LA.png	Cluster 60	6000 Bytes
…	…	…

Cluster 140

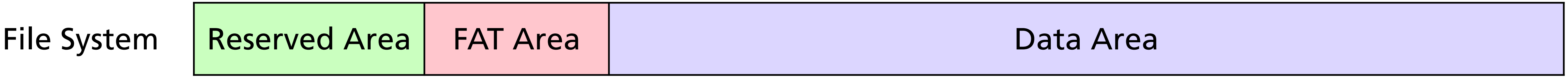
Name	Start	Size
.	Cluster 10	0
..	Cluster 110	0
0xe5ovie.mp4	Cluster 200	10 MB
…	…	…

Cluster 10

Files like these are
called "Orphan Files"

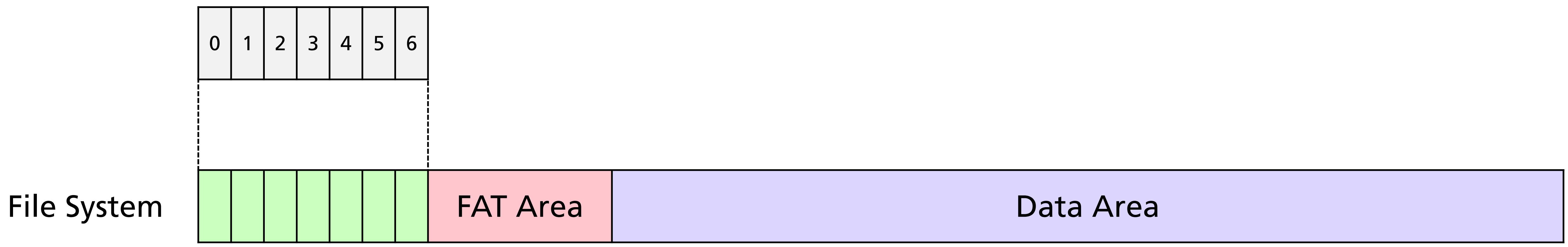
https://wiki.sleuthkit.org/index.php?title=Orphan_Files

FAT file system



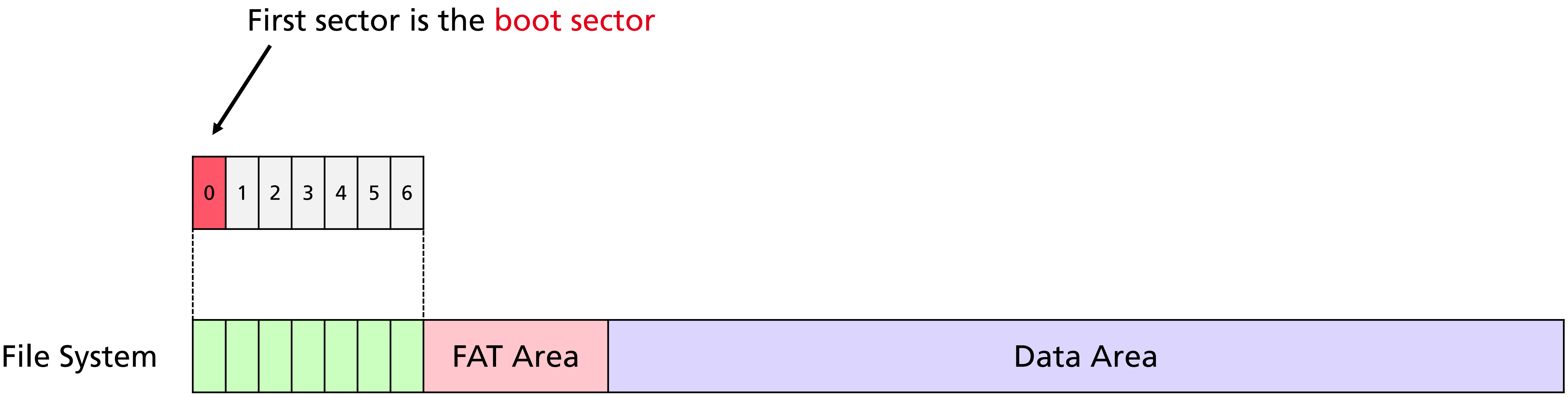
FAT file system

Reserved Area



FAT file system

Reserved Area

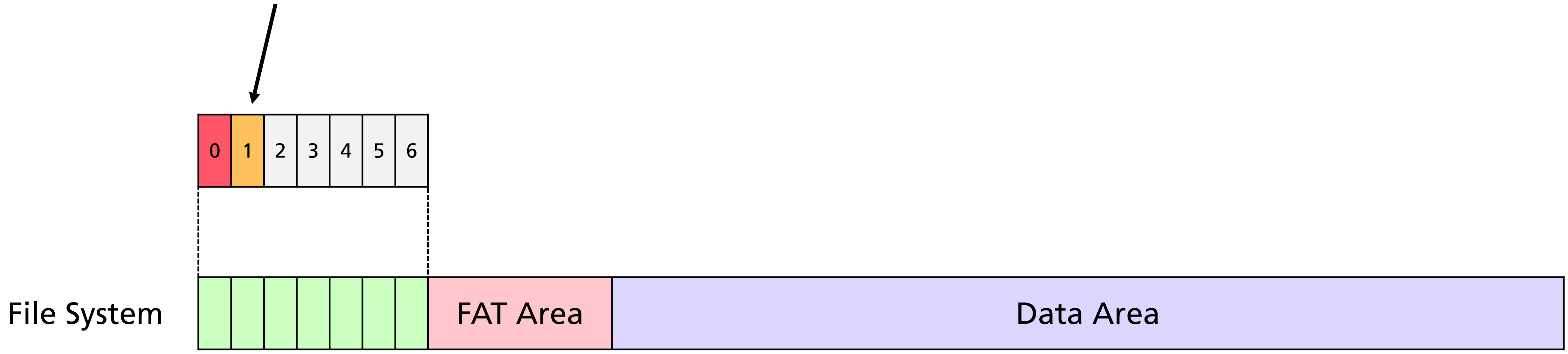


FAT file system

Reserved Area

For FAT32:

File System Information Sector (FSINFO)

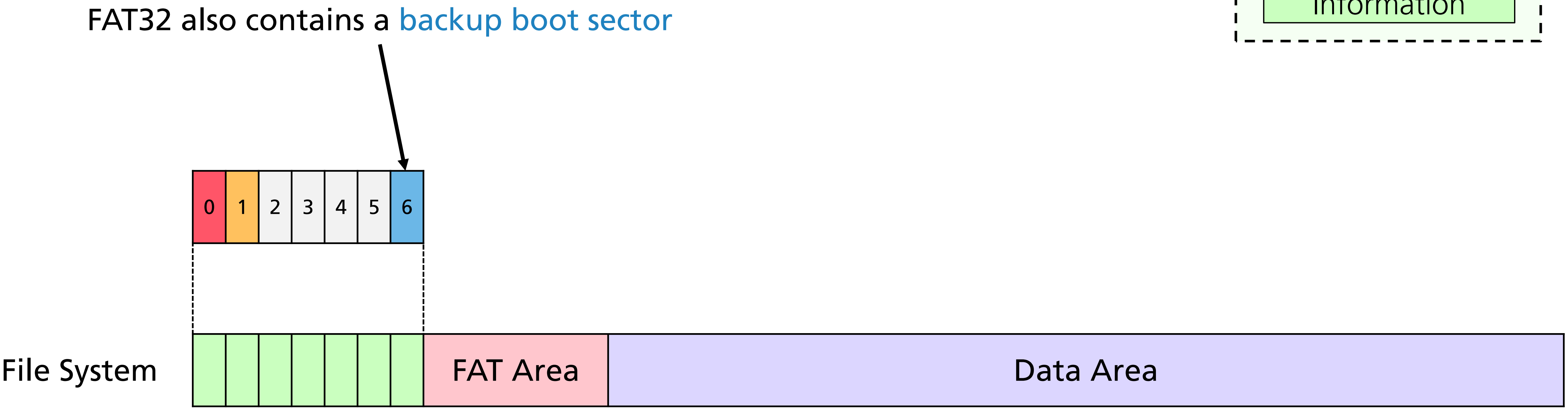


FAT file system

Reserved Area

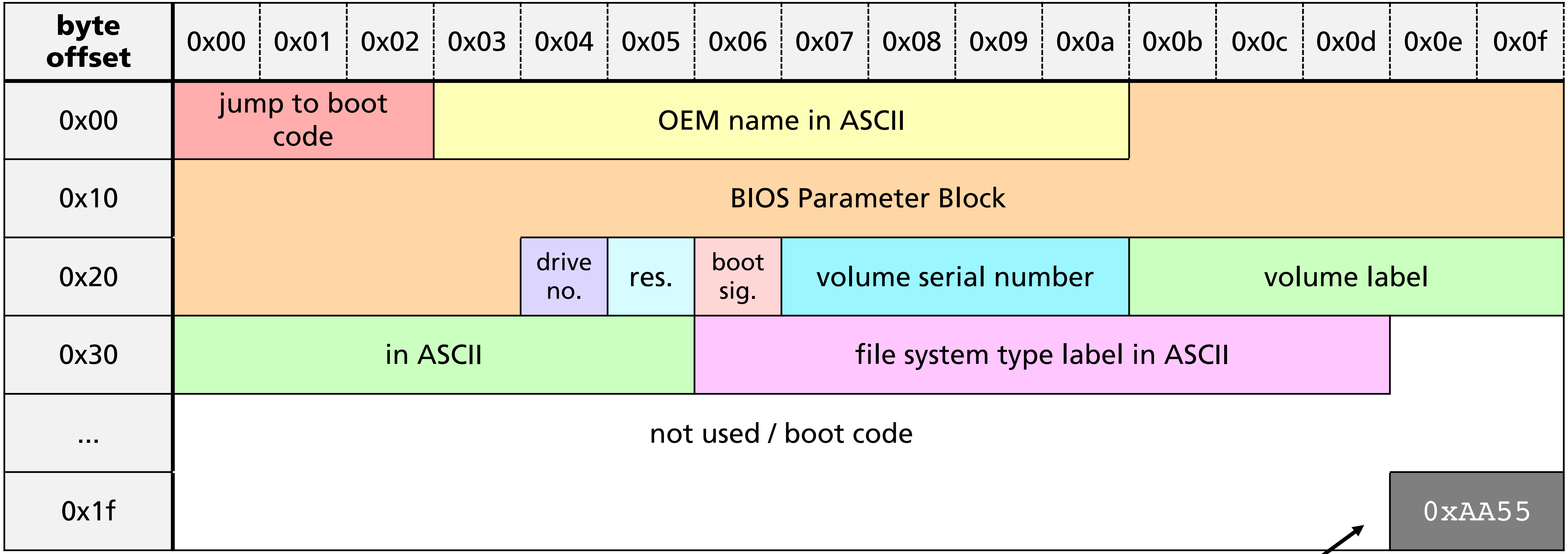
File System Category

Layout and Size Information



FAT file system

Boot Sector



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	jump to boot code			OEM name in ASCII												
0x10	BIOS Parameter Block															
0x20					drive no.	res.	boot sig.	volume serial number				volume label				
0x30	in ASCII						file system type label in ASCII									
...	not used / boot code															
0x1f															0xAA55	

Signature value at offset 510, which is usually also the end of the sector

FAT file system

Boot Sector

<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td colspan="3">jump to boot code</td><td colspan="8">OEM name in ASCII</td><td colspan="5"></td></tr>
<tr><td>0x10</td><td colspan="16">BIOS Parameter Block</td></tr>
<tr><td>0x20</td><td colspan="4"></td><td>drive no.</td><td>res.</td><td>boot sig.</td><td colspan="4">volume serial number</td><td colspan="5">volume label</td></tr>
<tr><td>0x30</td><td colspan="6">in ASCII</td><td colspan="8">file system type label in ASCII</td><td colspan="2"></td></tr>
<tr><td>...</td><td colspan="16">not used / boot code</td></tr>
<tr><td>0x1f</td><td colspan="14"></td><td colspan="2">0xAA55</td></tr>
</table>

Signature values are a great reference to look for in case of missing or damaged data (e.g. missing partition tables).

Signature Detection

sigfind

https://www.sleuthkit.org

sigfind(1) - Linux man page

Name

sigfind - Find a binary signature in a file

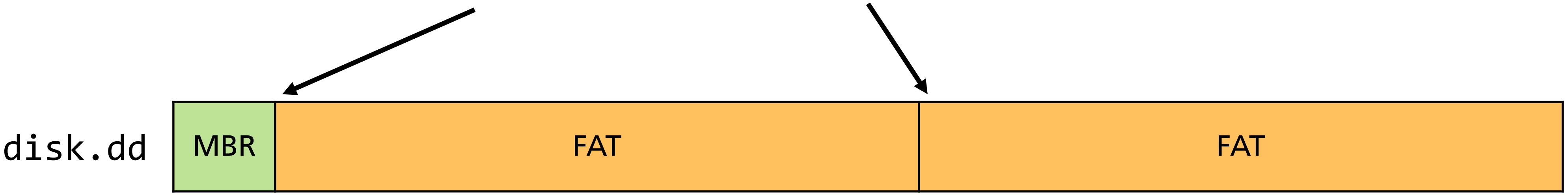
Synopsis

sigfind [-b bsize **] [-o** offset **] [-t** template **] [-lV] [** hex_signature **]** file

Description

sigfind searches through a file and looks for the hex_signature at a given offset. This can be used to search for lost boot sectors, superblocks, and partition tables.

Usually, the start of a file system can be inferred from the partition layout



Signature Detection

sigfind

https://www.sleuthkit.org

sigfind(1) - Linux man page

Name

sigfind - Find a binary signature in a file

Synopsis

sigfind [-b bsize **] [-o** offset **] [-t** template **] [-lV] [** hex_signature **]** file

Description

sigfind searches through a file and looks for the hex_signature at a given offset. This can be used to search for lost boot sectors, superblocks, and partition tables.

```
$ sigfind -o 510 -l AA55 disk.dd
```

Signature Detection

sigfind

https://www.sleuthkit.org

sigfind(1) - Linux man page

Name

sigfind - Find a binary signature in a file

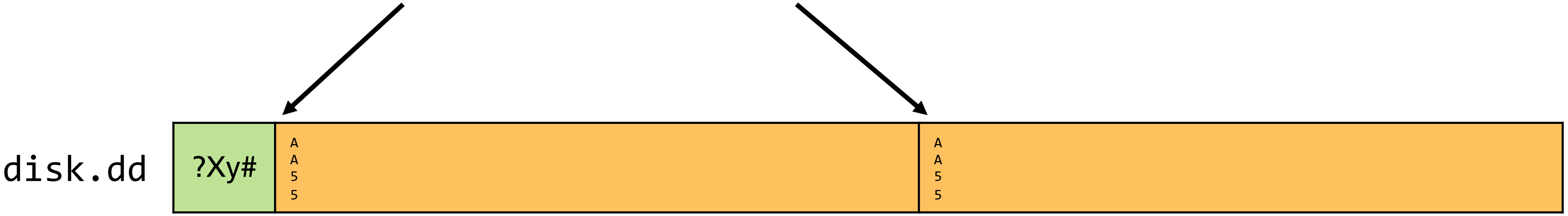
Synopsis

sigfind [-b bsize **] [-o** offset **] [-t** template **] [-lV] [** hex_signature **]** file

Description

sigfind searches through a file and looks for the hex_signature at a given offset. This can be used to search for lost boot sectors, superblocks, and partition tables.

```
$ sigfind -o 510 -l AA55 disk.dd
```



Signature Detection

gpart

gpart(8) - Linux man page

Name

gpart - guess PC-type hard disk partitions

Synopsis

```
gpart [options] device

       Options:  [-b <backup MBR>][-C c,h,s][-c][-d][-E][-e][-f] [-g][-h][-i]
                 [-K <last-sector>][-k <# of sectors>] [-L] [-l <log file>]
                 [-n <increment>] [-q][-s <sector-size>] [-t <module-name>][-V][-v]
                 [-W <device>][-w <module-name, weight>]
```

Description

gpart tries to guess which partitions are on a hard disk. If the primary partition table has been lost, overwritten or destroyed the partitions still exist on the disk but the operating system cannot access them.

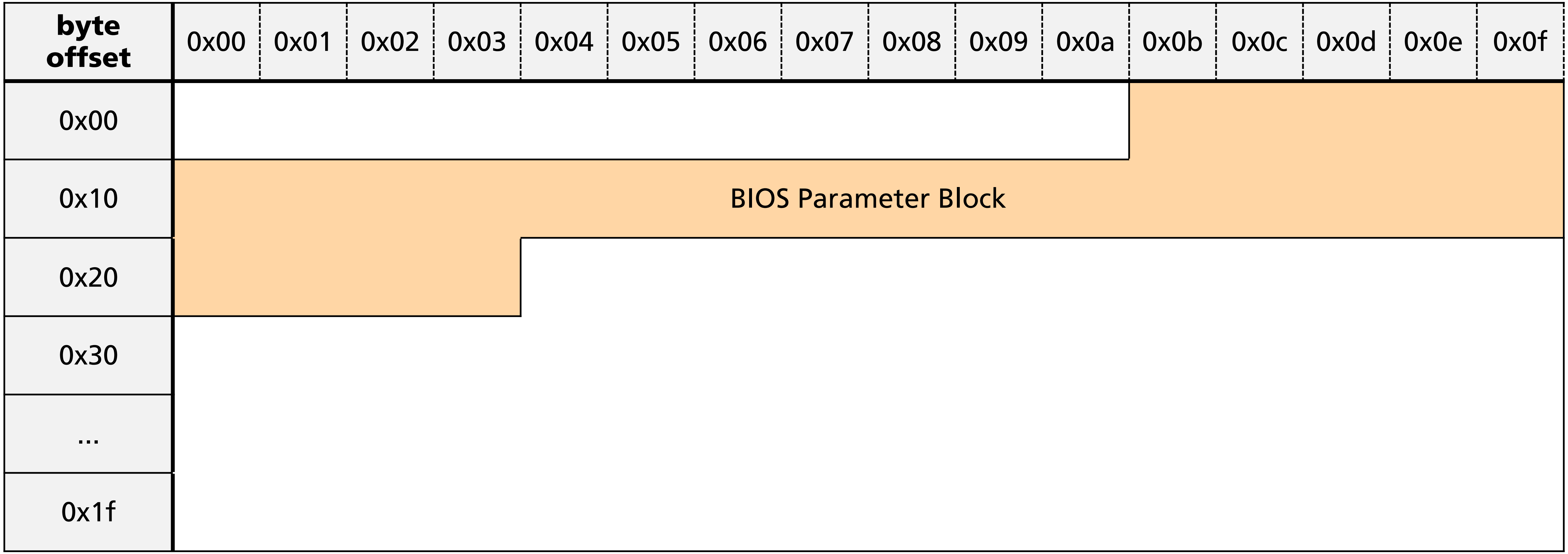
FAT file system

Boot Sector

<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td colspan="3">jump to boot code</td><td colspan="8">OEM name in ASCII</td><td colspan="5"></td></tr>
<tr><td>0x10</td><td colspan="16">BIOS Parameter Block</td></tr>
<tr><td>0x20</td><td colspan="4"></td><td>drive no.</td><td>res.</td><td>boot sig.</td><td colspan="4">volume serial number</td><td colspan="5">volume label</td></tr>
<tr><td>0x30</td><td colspan="6">in ASCII</td><td colspan="8">file system type label in ASCII</td><td colspan="2"></td></tr>
<tr><td>...</td><td colspan="16">not used / boot code</td></tr>
<tr><td>0x1f</td><td colspan="14"></td><td colspan="2">0xAA55</td></tr>
</table>

FAT file system

Boot Sector



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00																
0x10									BIOS Parameter Block							
0x20																
0x30																
...																
0x1f																

FAT file system

BIOS Parameter Block

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size				
0x10																
0x20																
0x30																
…																
0x1f																

Size of a sector in bytes (usually 512, 1024, 2048 or 4096)

```
mkfs.fat - create an MS-DOS FAT filesystem


-S LOGICAL-SECTOR-SIZE Specify the number of bytes per logical
sector. Must be a power of 2 and greater than or equal to 512, i.e.
512, 1024, 2048, 4096, 8192, 16384, or 32768. Values larger than
4096 are not conforming to the FAT file system specification and
may not work everywhere.
```

https://www.man7.org/linux/man-pages/man8/mkfs.fat.8.html

FAT file system

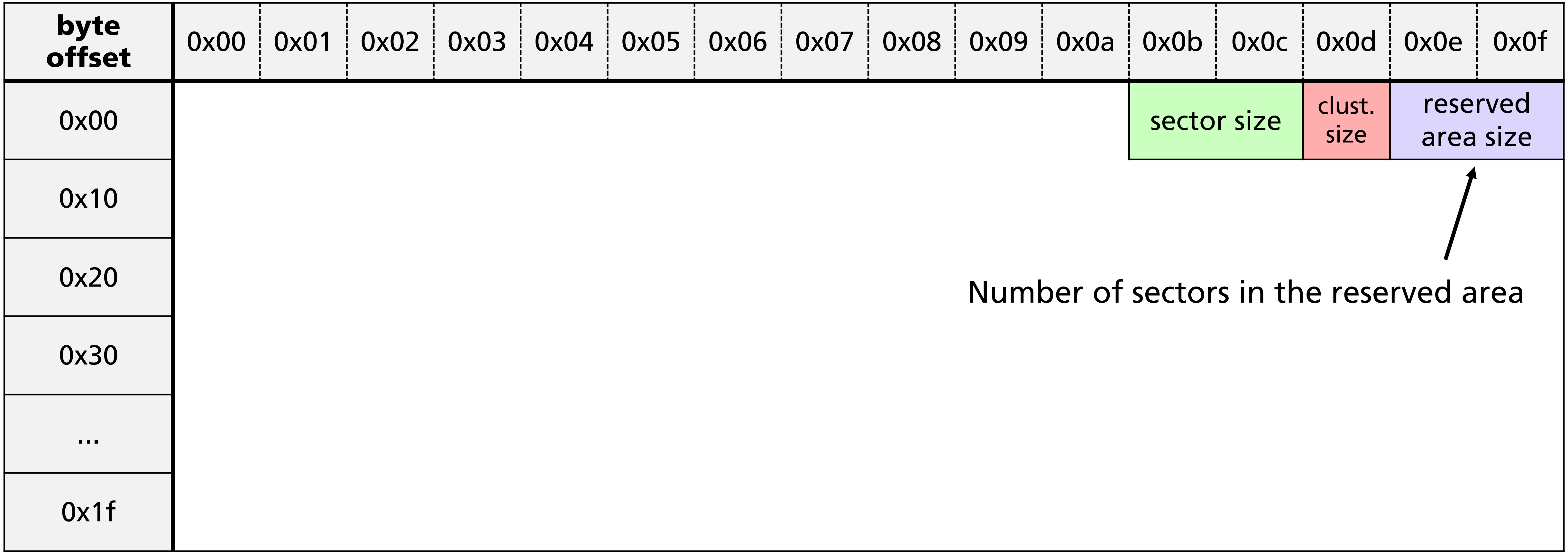
BIOS Parameter Block

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size		clust. size		
0x10																
0x20																
0x30																
...																
0x1f																

Sectors per cluster.
Value must be a power of 2 (i.e. 1, 2, 4, 8, 16, 32, 64, 128)

FAT file system

BIOS Parameter Block



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size		clust. size	reserved area size	
0x10																
0x20																
0x30																
...																
0x1f																

Number of sectors in the reserved area

FAT file system

BIOS Parameter Block

<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td colspan="11"></td><td colspan="2">sector size</td><td>clust. size</td><td colspan="2">reserved area size</td></tr>
<tr><td>0x10</td><td># of FATs</td><td colspan="2">max. # of root entries</td><td colspan="2">total sectors</td><td>media type</td><td colspan="2">FAT size in sectors</td><td colspan="4">geometry information</td><td colspan="4">hidden sectors</td></tr>
<tr><td>0x20</td><td colspan="4">total sectors (4 bytes)</td><td colspan="12"></td></tr>
<tr><td>0x30</td><td colspan="16"></td></tr>
<tr><td>...</td><td colspan="16"></td></tr>
<tr><td>0x1f</td><td colspan="16"></td></tr>
</table>

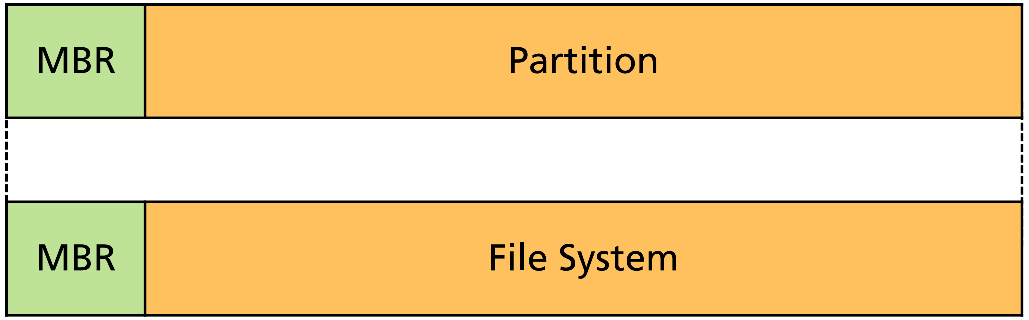
FAT file system

BIOS Parameter Block

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size		clust. size	reserved area size	
0x10	# of FATs	max. # of root entries		total sectors		media type	FAT size in sectors		geometry information				hidden sectors			
0x20	total sectors (4 bytes)															
0x30																
…																
0x1f																

FAT file system

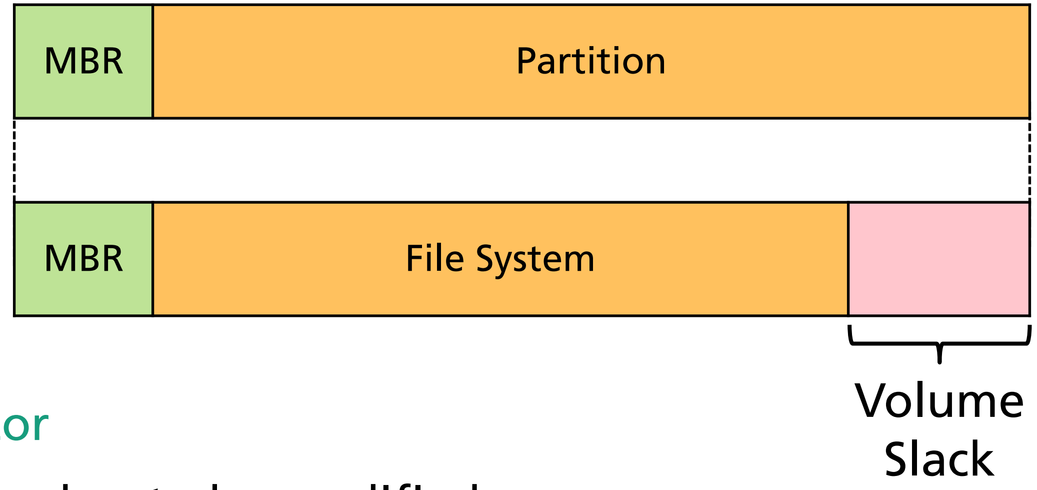
Analysis: Volume Slack



FAT file system

Analysis: Volume Slack

- Volume Slack describes the space between the end of a file system and the end of the volume it is stored on
- It can be detected by comparing the total size of a file system with the size of the volume (e.g. partition)
  - For FAT, this information is found in the boot sector
- Creating volume slack is easy, since only one parameter has to be modified



FAT file system

BIOS Parameter Block

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size		clust. size	reserved area size	
0x10	# of FATs	max. # of root entries		total sectors		media type	FAT size in sectors		geometry information				hidden sectors			
0x20	total sectors (4 bytes)															
0x30																
...																
0x1f																

FAT file system

BIOS Parameter Block

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size		clust. size	reserved area size	
0x10	# of FATs	max. # of root entries		total sectors		media type	FAT size in sectors		geometry information				hidden sectors			
0x20	total sectors (4 bytes)															
0x30																
…																
0x1f																

Number of sectors before the start of the partition the file system is using (beginning of the partition)

FAT file system

BIOS Parameter Block

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00												sector size		clust. size	reserved area size	
0x10	# of FATs	max. # of root entries		total sectors		media type	FAT size in sectors		geometry information				hidden sectors			
0x20	total sectors (4 bytes)															
0x30																
…																
0x1f																

File System Category

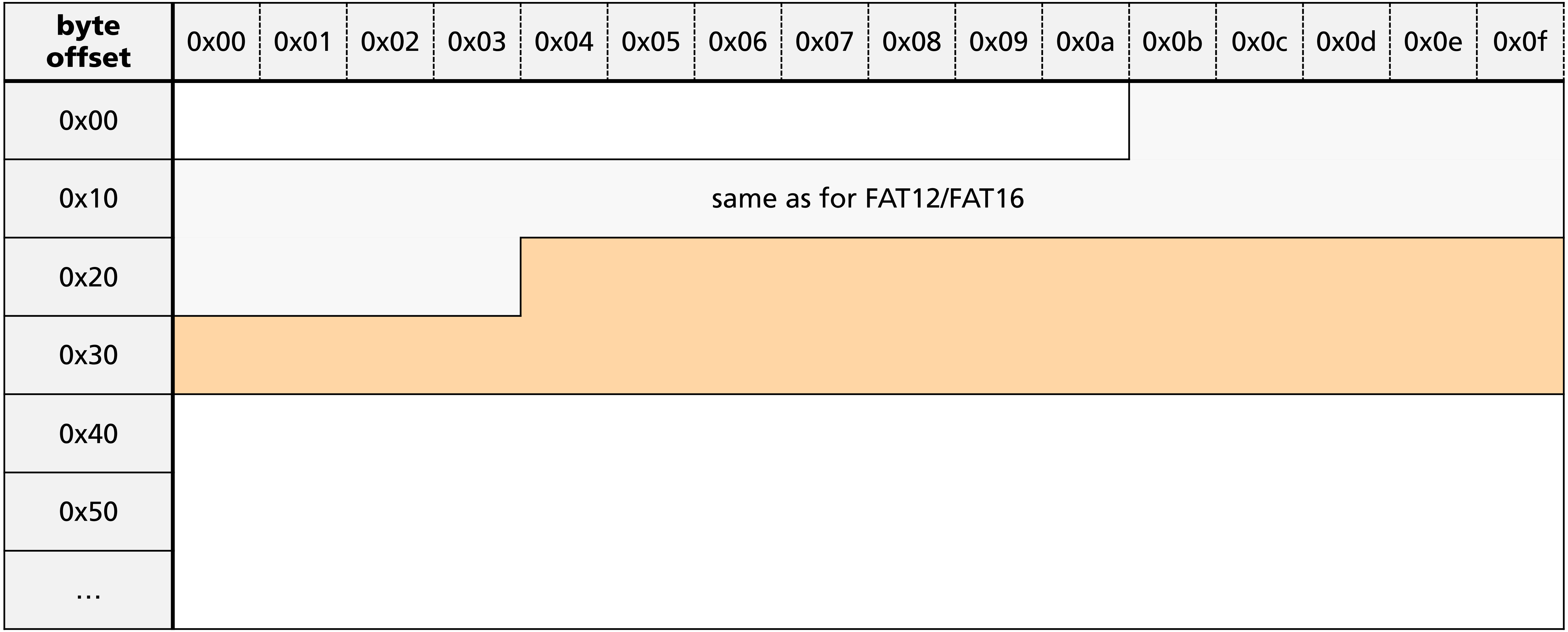
Layout and Size Information

File System

Reserved Area	FAT Area	Data Area

FAT file system

BIOS Parameter Block (FAT32)



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00																
0x10									same as for FAT12/FAT16							
0x20																
0x30																
0x40																
0x50																
…																

FAT file system

BIOS Parameter Block (FAT32)

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00																
0x10	same as for FAT12/FAT16															
0x20					FAT size in sectors				FAT usage flags		version		root directory cluster			
0x30	FSINFO sector		Backup boot sector		reserved											
0x40																
0x50																
…																

FAT file system

FSINFO

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x41615252															
…																
0x1e																
0x1f																

FSINFO sector signature

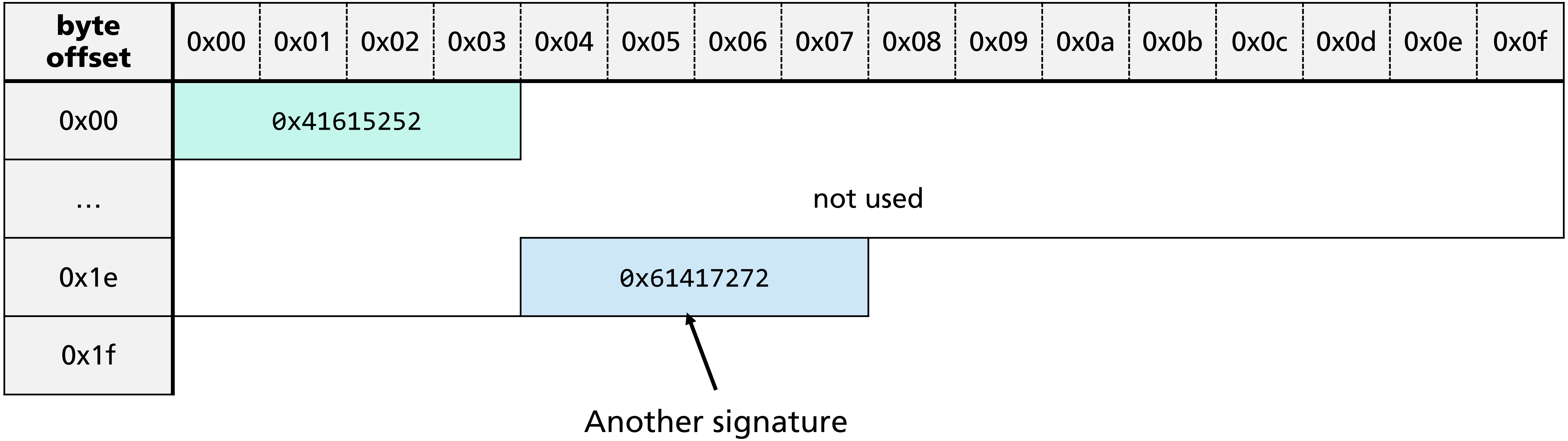
FAT file system

FSINFO

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x41615252				not used											
…																
0x1e																
0x1f																

FAT file system

FSINFO



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x41615252															
…	not used															
0x1e					0x61417272											
0x1f																

Another signature

FAT file system

FSINFO

<table>
<tr><th>byte offset</th><th>0x00</th><th>0x01</th><th>0x02</th><th>0x03</th><th>0x04</th><th>0x05</th><th>0x06</th><th>0x07</th><th>0x08</th><th>0x09</th><th>0x0a</th><th>0x0b</th><th>0x0c</th><th>0x0d</th><th>0x0e</th><th>0x0f</th></tr>
<tr><td>0x00</td><td colspan="4">0x41615252</td><td colspan="12"></td></tr>
<tr><td>…</td><td colspan="16">not used</td></tr>
<tr><td>0x1e</td><td colspan="4"></td><td colspan="4">0x61417272</td><td colspan="4">number of free clusters</td><td colspan="4"></td></tr>
<tr><td>0x1f</td><td colspan="16"></td></tr>
</table>

Last number of free clusters (which must not be correct).
`0xFFFFFFFF` indicates an unknown value.

FAT file system

FSINFO

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x41615252															
…	not used															
0x1e					0x61417272				number of free clusters				next free cluster			
0x1f																

Usually set to the last allocated cluster, indicating from where to search for other free clusters.

FAT file system

FSINFO

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x41615252															
…	not used															
0x1e					0x61417272				number of free clusters				next free cluster			
0x1f	reserved															

FAT file system

FSINFO

byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	0x41615252															
…	not used															
0x1e					0x61417272				number of free clusters				next free cluster			
0x1f	reserved												0xAA550000			

Another signature

The Sleuth Kit

fsstat

https://www.sleuthkit.org

fsstat(1) - Linux man page

Name

fsstat - Display general details of a file system

Synopsis

fsstat [-f fstype] [-i imgtype] [-o imgoffset] [-b dev_sector_size] [-tvV] image [images]

Description

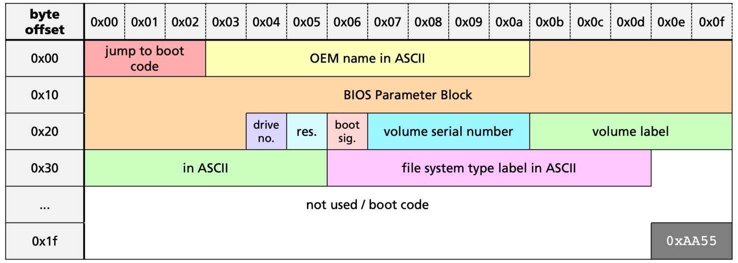
fsstat displays the details associated with a file system. The output of this command is file system specific. At a minimum, the range of meta-data values (inode numbers) and content units (blocks or clusters) are given. Also given are details from the Super Block, such as mount times and and features. For file systems that use groups (FFS and EXT2FS), the layout of each group is listed. For a FAT file system, the FAT table is displayed in a condensed format. Note that the data is in sectors and not in clusters.

The Sleuth Kit

fsstat

FAT file system

Boot Sector



byte offset	0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0a	0x0b	0x0c	0x0d	0x0e	0x0f
0x00	jump to boot code			OEM name in ASCII												
0x10	BIOS Parameter Block															
0x20					drive no.	res.	boot sig.	volume serial number				volume label				
0x30	in ASCII						file system type label in ASCII									
...	not used / boot code															
0x1f															0xAA55	

```
$ fsstat disk.dd -o 2048

FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT16

OEM Name: mkfs.fat
Volume ID: 0x3075b80d
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory):
File System Type Label: FAT16

Sectors before file system: 0

File System Layout (in sectors)
Total Range: 0 – 202751
* Reserved: 0 – 3
** Boot Sector: 0
* FAT 0: 4 – 203
* FAT 1: 204 – 403
* Data Area: 404 – 202751
** Root Directory: 404 – 435
** Cluster Area: 436 - 202751
```

```
METADATA INFORMATION
--------------------------------------------
Range: 2 – 3237574
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 2048
Total Cluster Range: 2 - 50580

FAT CONTENTS (in sectors)

--------------------------------------------
```

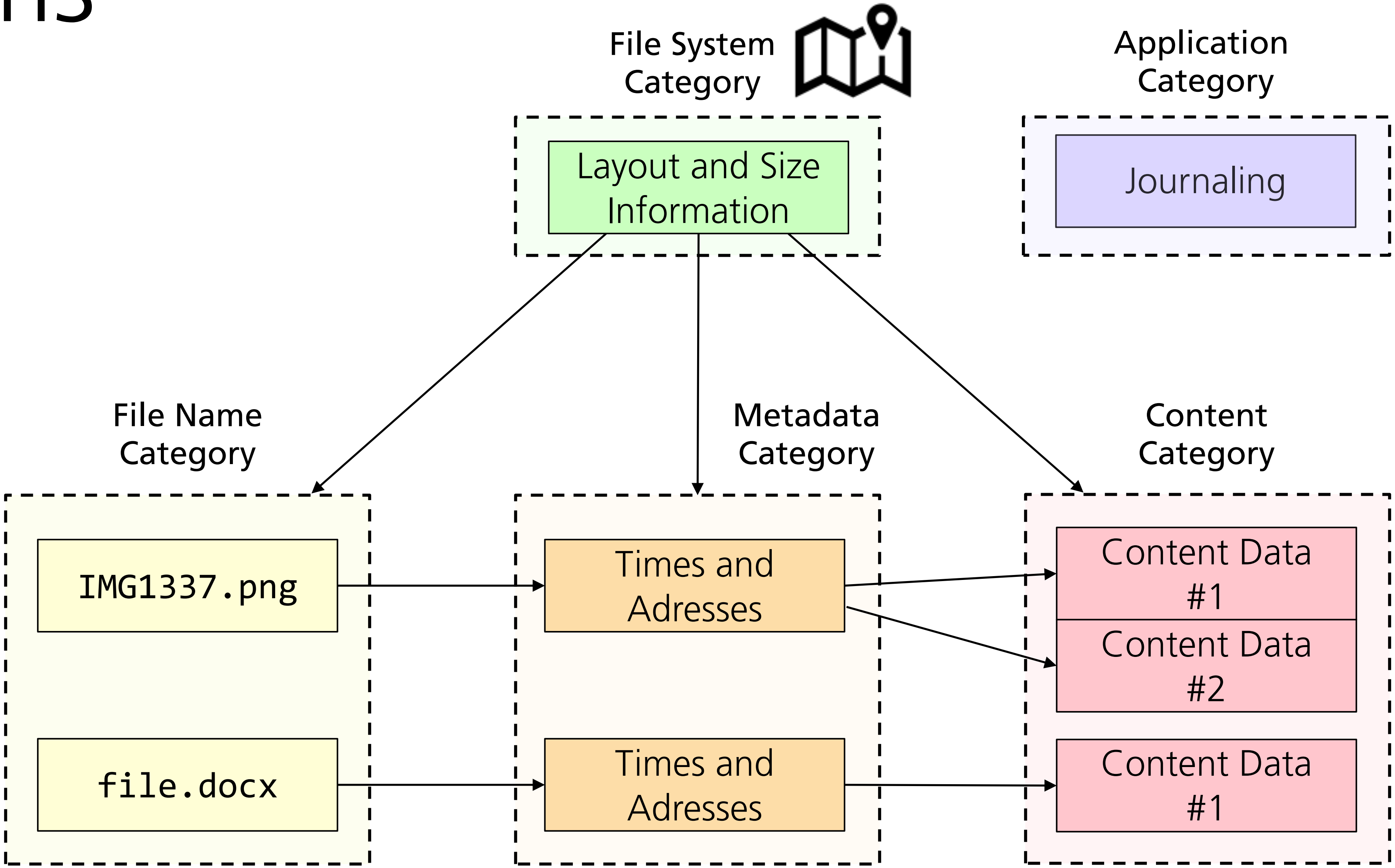
https://www.sleuthkit.org

File System Category

Layout and Size Information

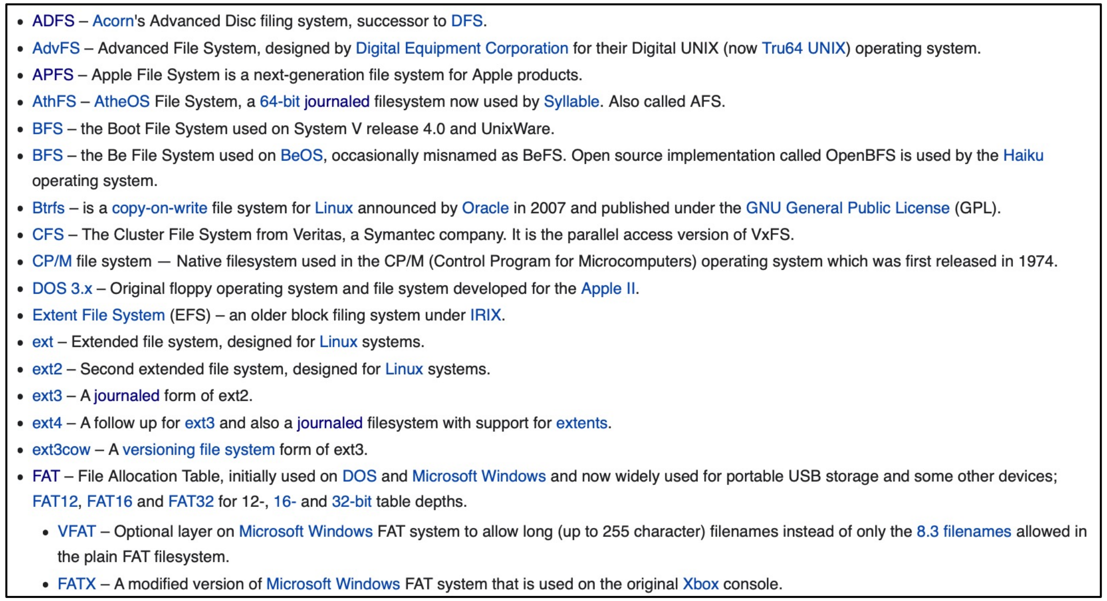
File Systems

Dr. Brian Carrier

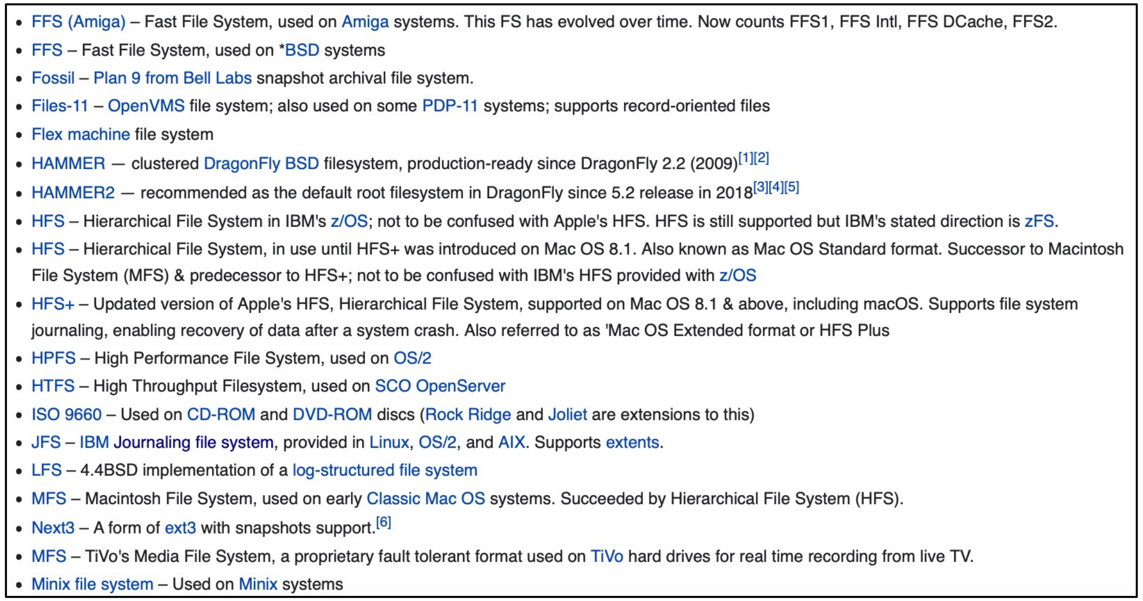


File Systems

- File system analysis including its results is completely dependent on the file system
- In most cases new file systems require new tools in order to achieve the most of it
- We have covered an important example, but there are more…



- ADFS – Acorn's Advanced Disc filing system, successor to DFS.
- AdvFS – Advanced File System, designed by Digital Equipment Corporation for their Digital UNIX (now Tru64 UNIX) operating system.
- APFS – Apple File System is a next-generation file system for Apple products.
- AthFS – AtheOS File System, a 64-bit journaled filesystem now used by Syllable. Also called AFS.
- BFS – the Boot File System used on System V release 4.0 and UnixWare.
- BFS – the Be File System used on BeOS, occasionally misnamed as BeFS. Open source implementation called OpenBFS is used by the Haiku operating system.
- Btrfs – is a copy-on-write file system for Linux announced by Oracle in 2007 and published under the GNU General Public License (GPL).
- CFS – The Cluster File System from Veritas, a Symantec company. It is the parallel access version of VxFS.
- CP/M file system — Native filesystem used in the CP/M (Control Program for Microcomputers) operating system which was first released in 1974.
- DOS 3.x – Original floppy operating system and file system developed for the Apple II.
- Extent File System (EFS) – an older block filing system under IRIX.
- ext – Extended file system, designed for Linux systems.
- ext2 – Second extended file system, designed for Linux systems.
- ext3 – A journaled form of ext2.
- ext4 – A follow up for ext3 and also a journaled filesystem with support for extents.
- ext3cow – A versioning file system form of ext3.
- FAT – File Allocation Table, initially used on DOS and Microsoft Windows and now widely used for portable USB storage and some other devices; FAT12, FAT16 and FAT32 for 12-, 16- and 32-bit table depths.
  - VFAT – Optional layer on Microsoft Windows FAT system to allow long (up to 255 character) filenames instead of only the 8.3 filenames allowed in the plain FAT filesystem.
  - FATX – A modified version of Microsoft Windows FAT system that is used on the original Xbox console.



- FFS (Amiga) – Fast File System, used on Amiga systems. This FS has evolved over time. Now counts FFS1, FFS Intl, FFS DCache, FFS2.
- FFS – Fast File System, used on *BSD systems
- Fossil – Plan 9 from Bell Labs snapshot archival file system.
- Files-11 – OpenVMS file system; also used on some PDP-11 systems; supports record-oriented files
- Flex machine file system
- HAMMER — clustered DragonFly BSD filesystem, production-ready since DragonFly 2.2 (2009)[1][2]
- HAMMER2 — recommended as the default root filesystem in DragonFly since 5.2 release in 2018[3][4][5]
- HFS – Hierarchical File System in IBM's z/OS; not to be confused with Apple's HFS. HFS is still supported but IBM's stated direction is zFS.
- HFS – Hierarchical File System, in use until HFS+ was introduced on Mac OS 8.1. Also known as Mac OS Standard format. Successor to Macintosh File System (MFS) & predecessor to HFS+; not to be confused with IBM's HFS provided with z/OS
- HFS+ – Updated version of Apple's HFS, Hierarchical File System, supported on Mac OS 8.1 & above, including macOS. Supports file system journaling, enabling recovery of data after a system crash. Also referred to as 'Mac OS Extended format or HFS Plus
- HPFS – High Performance File System, used on OS/2
- HTFS – High Throughput Filesystem, used on SCO OpenServer
- ISO 9660 – Used on CD-ROM and DVD-ROM discs (Rock Ridge and Joliet are extensions to this)
- JFS – IBM Journaling file system, provided in Linux, OS/2, and AIX. Supports extents.
- LFS – 4.4BSD implementation of a log-structured file system
- MFS – Macintosh File System, used on early Classic Mac OS systems. Succeeded by Hierarchical File System (HFS).
- Next3 – A form of ext3 with snapshots support.[6]
- MFS – TiVo's Media File System, a proprietary fault tolerant format used on TiVo hard drives for real time recording from live TV.
- Minix file system – Used on Minix systems

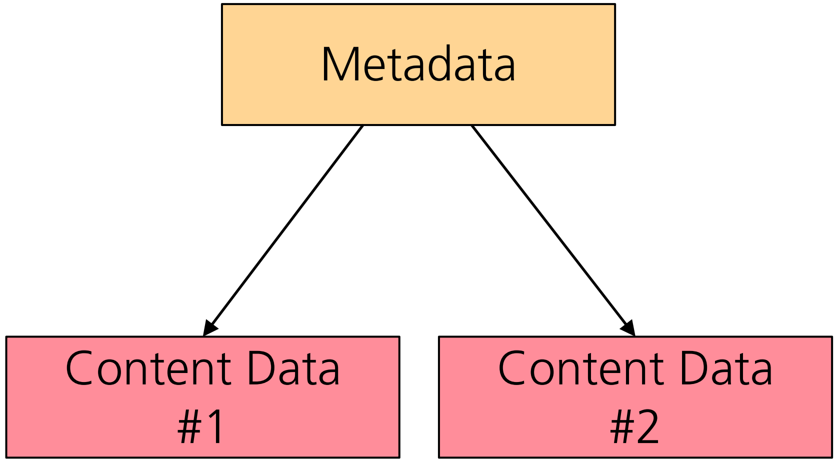
`https://en.wikipedia.org/wiki/List_of_file_systems` (part of)

Fancy File System Techniques

Copy-on-write

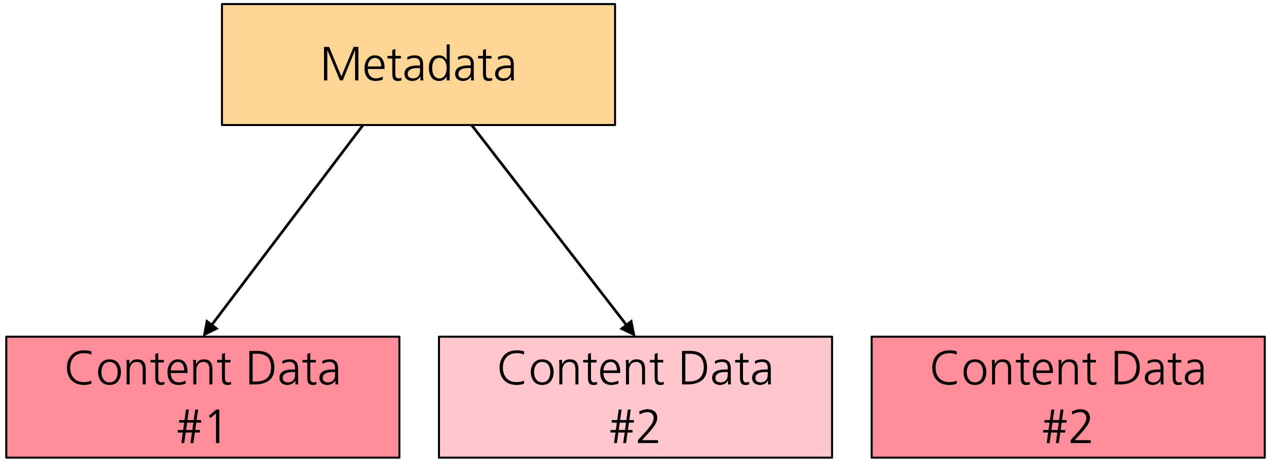
Copy-on-write

- Technique to ensure a file system stays in a consistent state



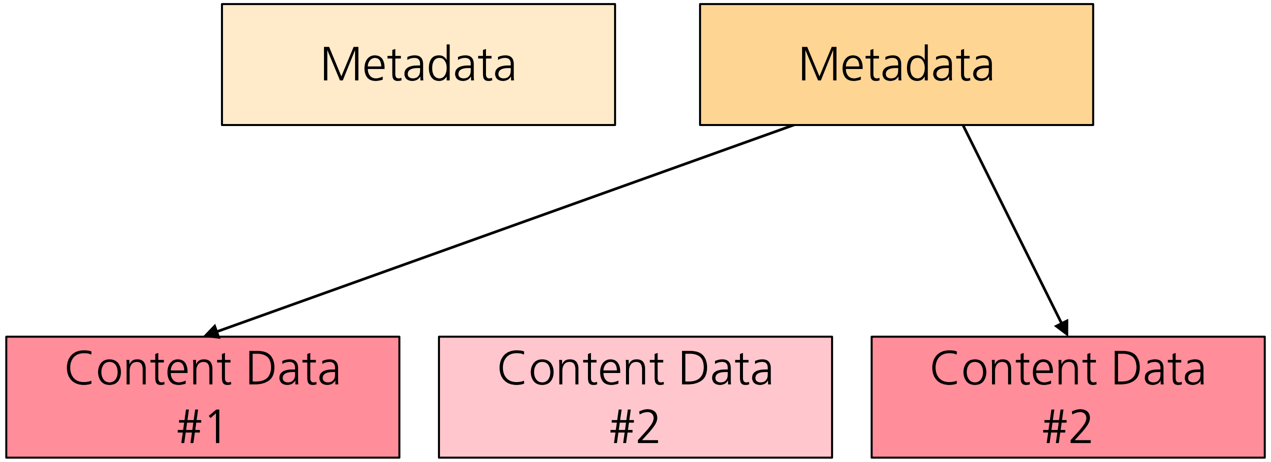
Copy-on-write

- Technique to ensure a file system stays in a consistent state
  - Content in the second data block is modified, so it is completely rewritten (copied) to a different area



Copy-on-write

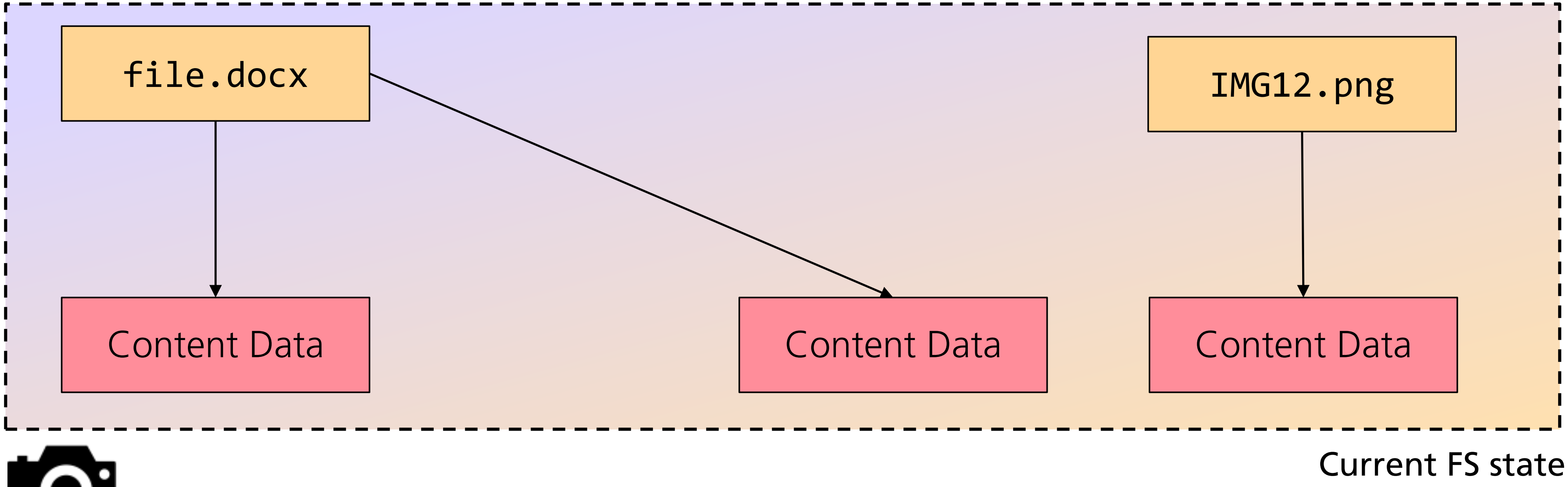
- Technique to ensure a file system stays in a consistent state
  - Same applies to metadata structures which are updated during this process (e.g. timestamps, addresses)



Results in a lot of artifacts!

Copy-on-write

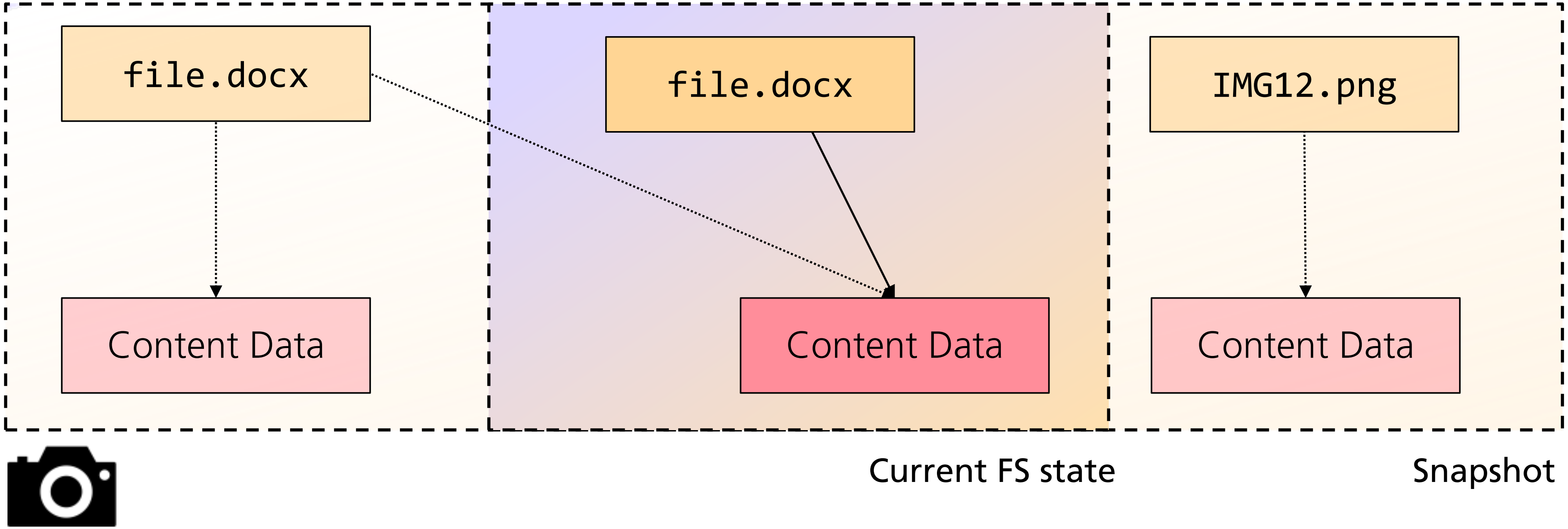
- Technique to ensure a file system stays in a consistent state
  - One possibility to ease the creation of snapshots



Copy-on-write

Possibility to go back in time!

- Technique to ensure a file system stays in a consistent state
  - One possibility to ease the creation of snapshots



Fancy File System Techniques

Copy-on-write

Snapshots

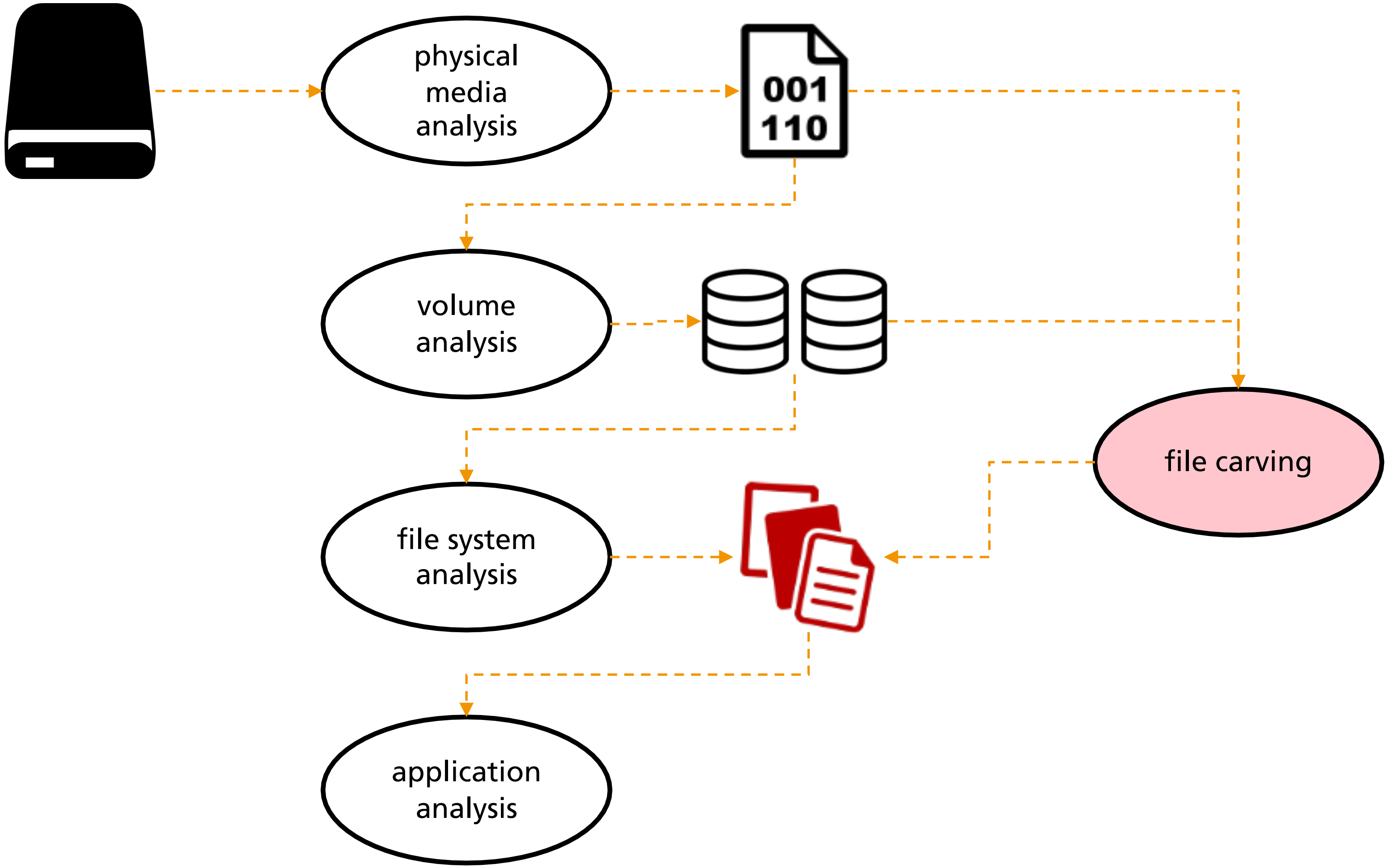
Encryption

Remote Conncetions

Support for multiple disks out of the box

File System Analysis

What if we cannot use the file system in order to obtain our artifacts?



Any Questions?