Deleted File Recovery. Part 2.

Understanding how it works and when it works

Motivation of Part 2

- In Part 1, we used the Autopsy tool on a Windows computer to recover some of the deleted files.
  - The Autopsy tool is GUI-based and easy to use.
  - We were able to recover many deleted files.
  - However, some questions remain to be answered.
- In Part 2, we go deeper to get answer to the questions.
  - We use command-line interface on a Linux system
    - Because that takes us much close to what happens under the hood.
    - Command-line is cool; don’t be scared!
  - Basically, we use a Kali Linux virtual machine on a Windows laptop.
    - Note that Kali VM is free and readily available to download.

Outline of Part 2

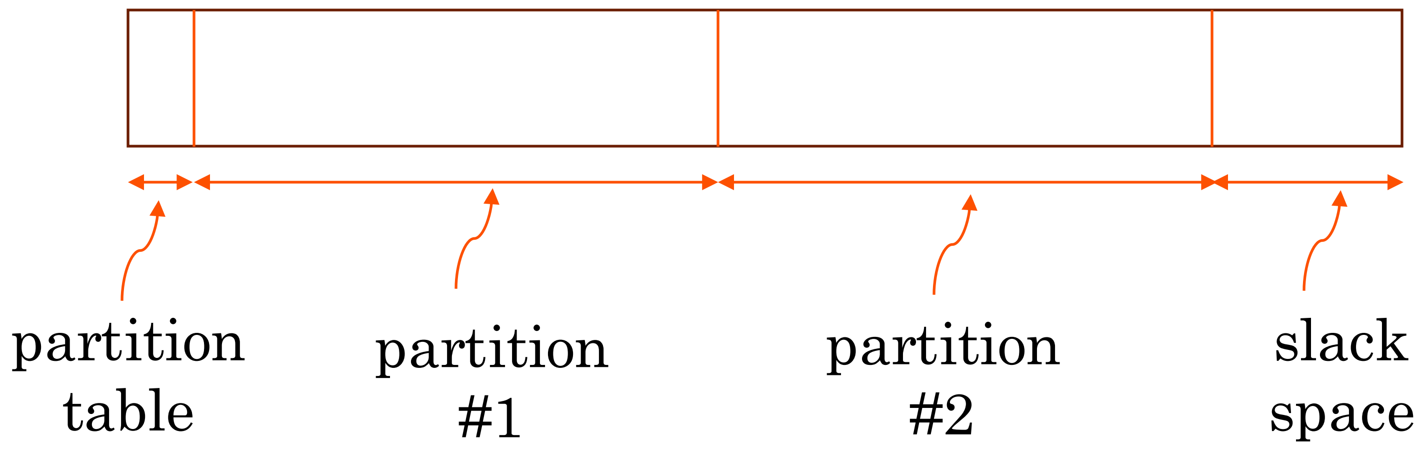
A. Thumb drive and partitions
B. Building file system in a partition
C. Basic design of FAT file system
  - Metadata vs. Real data
D. Deleted file recovery mechanisms
  - Metadata-based
  - File carving
E. A hands-on case study

How do we store data in a computer

- Storage medium
    - Hard disk (typically inside a computer)
    - USB thumb drive (a small portable device)
- Regular user's perspective
    - We can create one or more folders in the storage medium
    - We can put one or more files in a folder

Partitions in a thumb drive

- A thumb drive (or hard disk) may contain one or more partitions
  - As an example, a 128GB thumb drive can have two partitions each of size 64GB.
  - Analogy: A thumb drive is like a stretch of land owned by one or more owners. A partition is the part of land, which is under one person's ownership. Two neighboring partitions are separated by a clear boundary.
  - There is a small partition table at the beginning of the thumb drive
  - There can be some unused space known as slack space.

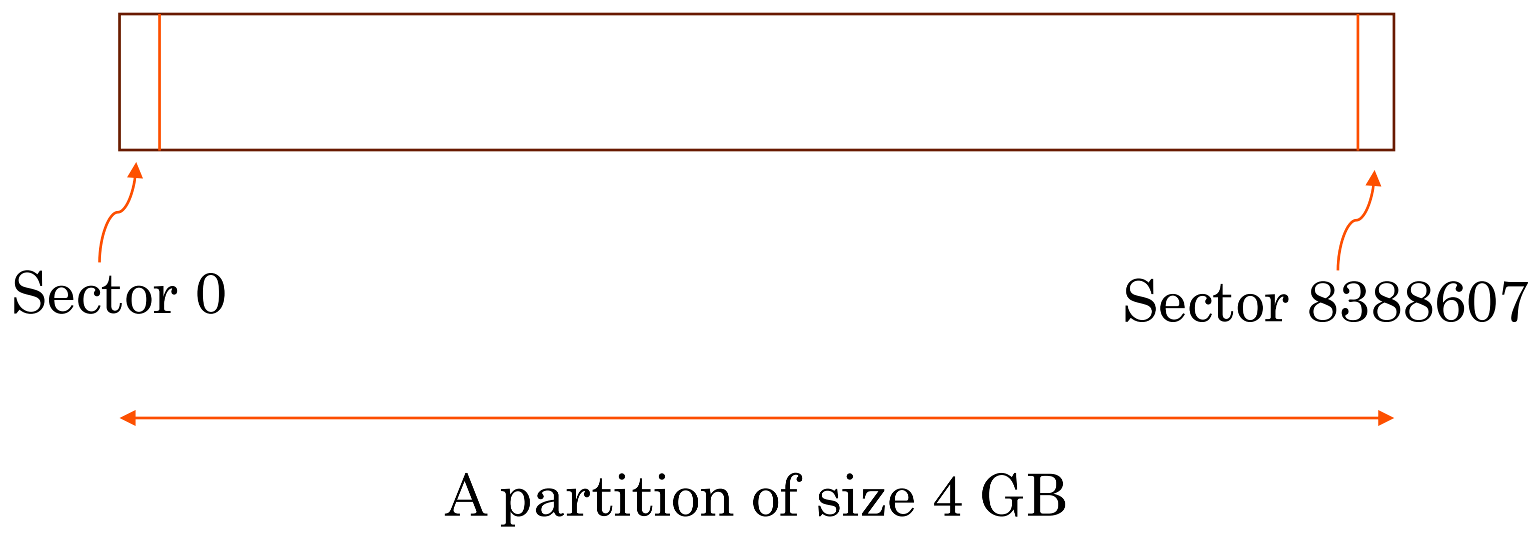


Partition vs. file system

- A bare partition is not yet ready to host files. To host files, we need to build a file system in the partition.
  - A file system has mechanism to host files or folders
  - FAT is an example file system, which is typically used in thumb drives
- Analogy:
  - A partition is like a piece of land whereas a file system is like the house built on that land.
  - Unless the file system (aka. house) is built, we cannot really store an item (like a folder with files in it) in the partition (aka. piece of land).
  - The house can have multiple bookshelves to host one or more folders.
  - A folder can contain one or more files.

Layout of a partition

- A partition is divided into sectors
  - Each sector is typically 512 byte
  - Each sector has an index. As an example, a 4GB partition has 8388608 sectors (as 4*1024*1024*1024 byte / 512 byte = 8388608): Sector 0 to Sector 8388607.



Outline of the current module

A. Thumb drive and partitions

B. Building file system in a partition

C. Basic design of FAT file system
   - Metadata vs. Real data

D. Deleted file recovery mechanisms
   - Metadata-based
   - File carving

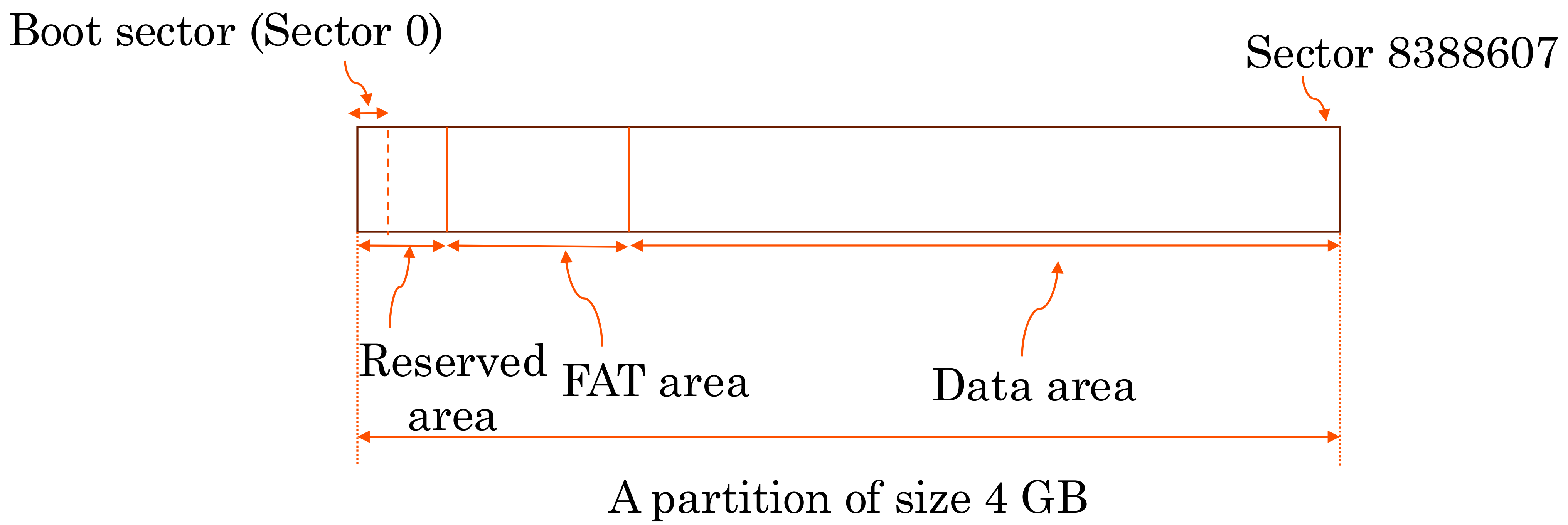
E. A hands-on case study

Building FAT file system on a partition

- As we already know, building a file system is like building a house on a piece of land (aka. partition)
- At the entrance of the house, we put a map (with directions) so that a visitor can find what is where
- Similarly, the first part of a FAT file system, called reserved area, has basic information about the file system, e.g., the location of FAT area, location of Data area, size of a cluster, and more.
    - The first sector in reserved area is called Boot Sector; among other things it tells us the length of the reserved area.

Building FAT file system on a partition (contd.)

- Data area is the part of the file system, which hosts the content of ALL files and folders
- A cluster is the minimum unit of Data area; typically, 8 sectors make one cluster
- FAT area contains (two copies of) a table (called FAT table) that tells us which clusters are chained together to host a file; it also tells us whether a cluster is unallocated (i.e., not hosting an active file)



Outline of the current module

A. Thumb drive and partitions

B. Building file system in a partition

C. Basic design of FAT file system

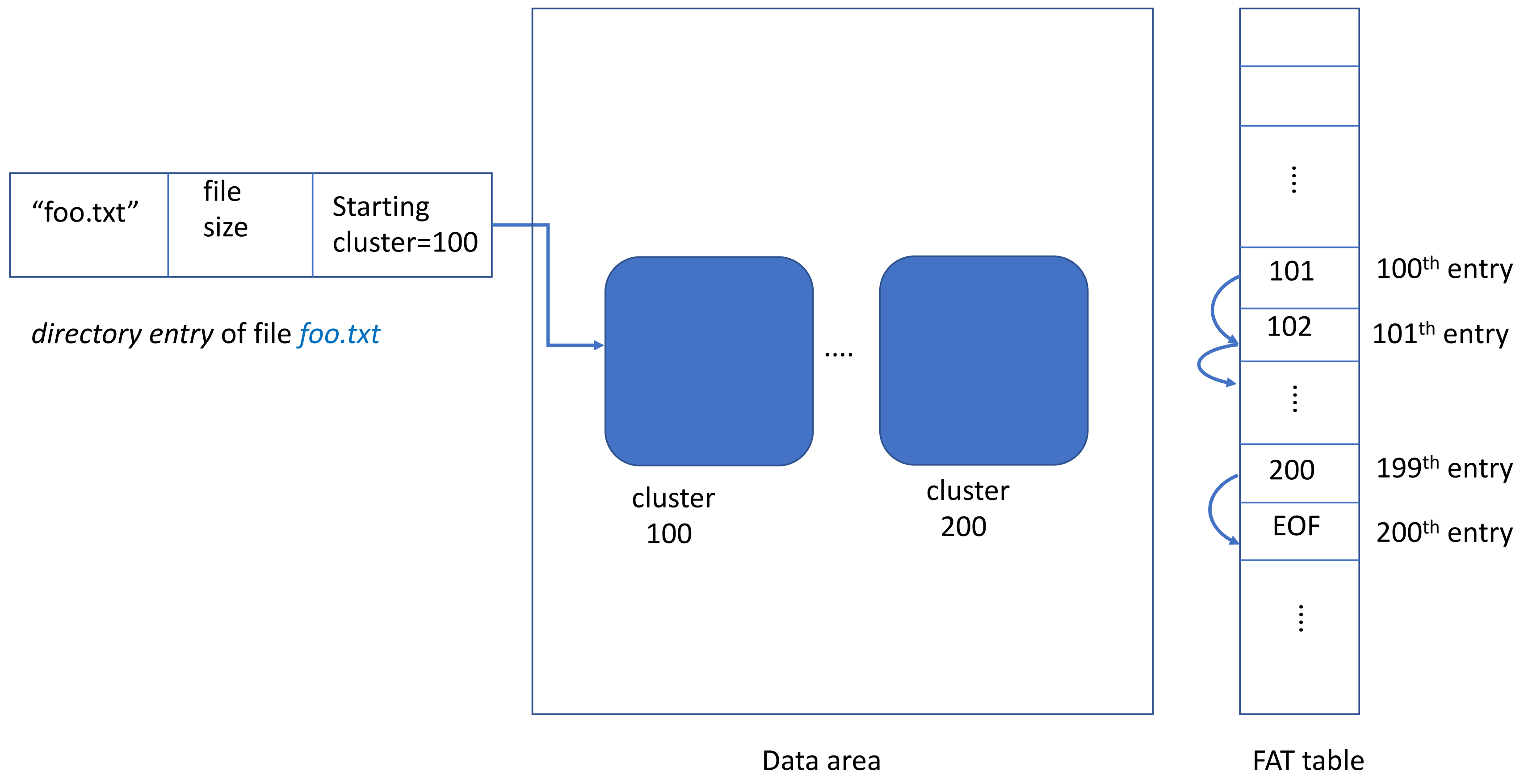
- Metadata vs. Real data

D. Deleted file recovery mechanisms

- Metadata-based
- File carving

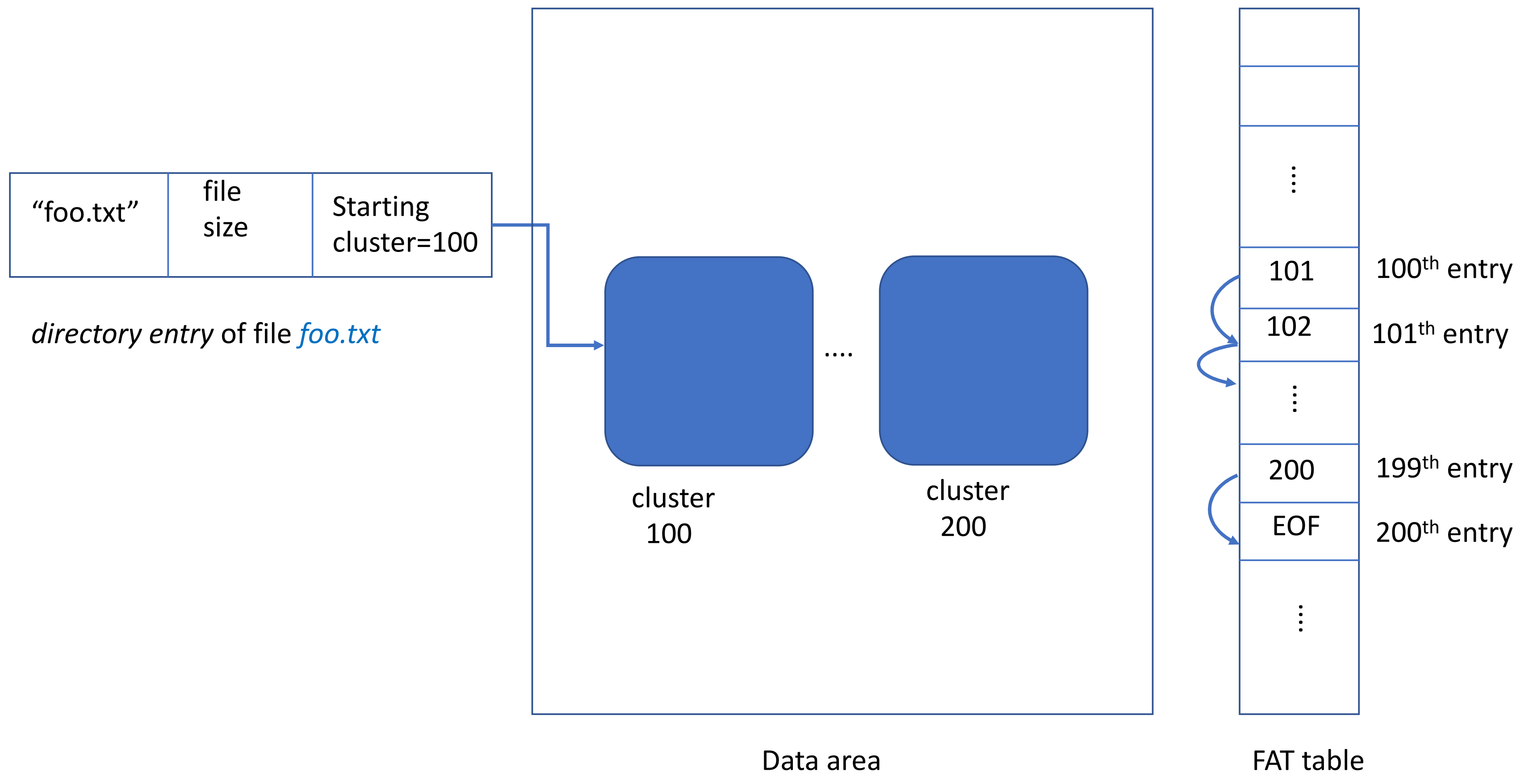
E. A hands-on case study

How a file is implemented in FAT file system



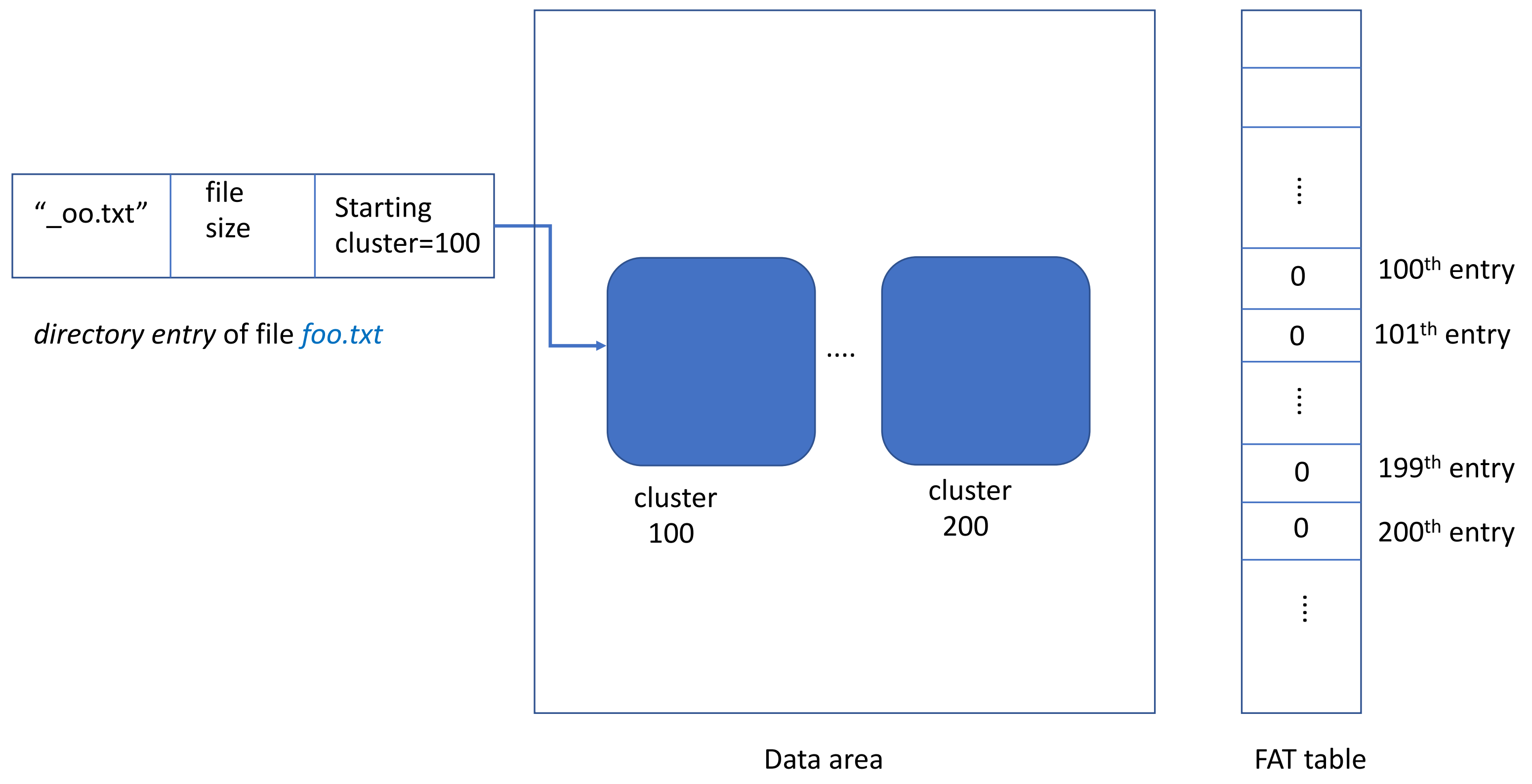
- The above figure shows the implementation of a file, foo.txt
- The real data of foo.txt sits in a chain of clusters: 100, 101, …, and 200
- The FAT table implements this chain
  - 100th entry of the table is 101, which means cluster 101 comes next in the chain after cluster 100
- There is a piece (32 bytes) of metadata, called directory entry, for foo.txt

How a file is implemented in FAT file system (contd.)



- Again, note that foo.txt has real data and metadata known as directory entry
- The directory entry of foo.txt has 3 main items: filename, file size, and the index of the starting cluster (which is 100)
- The last entry of foo.txt in FAT table is "EOF", which tells us foo.txt ends at cluster 200.
- If you are wondering about where the directory entry is located
  - The parent directory of foo.txt hosts foo.txt's directory entry.

What happens when foo.txt is deleted?



- The directory entry (aka. metadata) remains mostly intact
  - Only the first character of the file name is replaced by '_', which is an indicator that this directory entry is "free" now to be assigned to other files, if necessary
- The real data of foo.txt also remains
  - Only the corresponding entries in the FAT table become zero, which indicates that the corresponding clusters are "free" now to be allocated to other files, if necessary
- Summary: Unless new files are created in the system after the "deletion" operation, both metadata and real data of the deleted file remain. That means we can leverage the metadata to recover the deleted file.

Outline of the current module

A. Thumb drive and partitions
B. Building file system in a partition
C. Basic design of FAT file system
  - Metadata vs. Real data
D. Deleted file recovery mechanisms
  - Metadata-based
  - File carving
E. A hands-on case study

Mechanisms to recover a deleted file

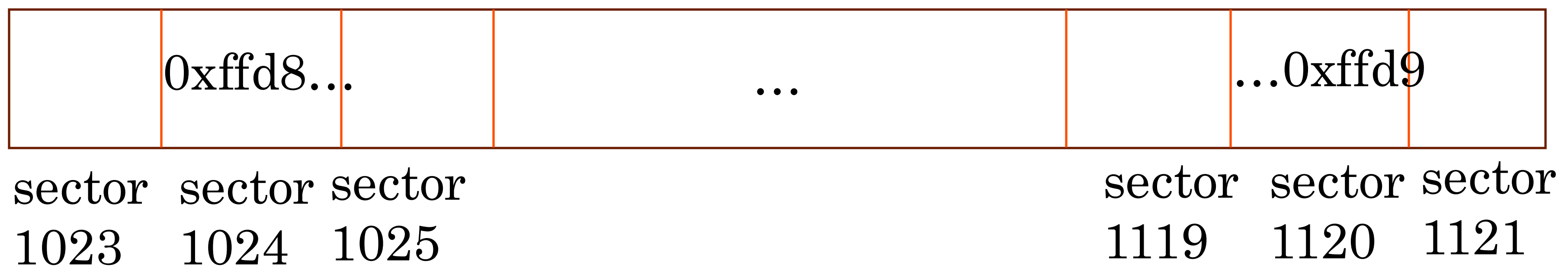
There are two mechanisms

- Metadata-based recovery
  - We can leverage this mechanism if the metadata is still available. Later we present a hands-on case study to demonstrate this.
  - However, if the metadata is deleted or overwritten, this mechanism is useless.
- File carving
  - This mechanism works only if the real data (of the deleted file) has a start signature and an end signature
  - Only certain file types (e.g., pdf, jpg, etc.) have such signatures. As an example, text file does not have such signature, so file carving cannot recover a text file

File Carving

- In "*data carving*" process, we try to find the start signature and end signature of a file in an unallocated data chunk.
  - For instance, Autopsy tool can do data carving.
  - Many other tools (photorec, scalpel, foremost) are out there

An example of File Carving



- The start signature (ffd8) and end signature (ffd9) of a JPEG file are searched in a data chunk.
- The output is a carved file (from the start point in Sector 1024 to the end point in Sector 1120).

Outline of the current module

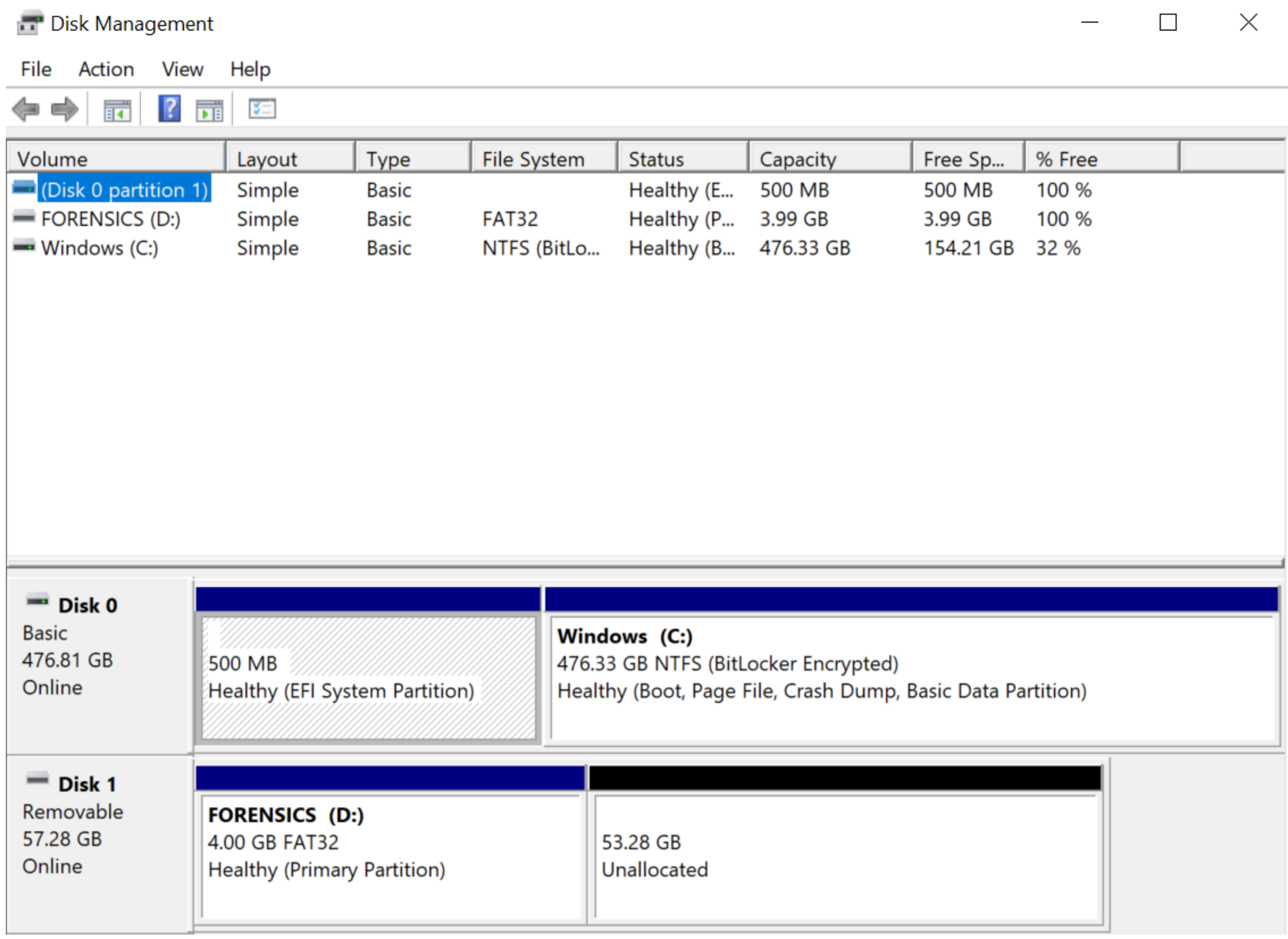
A. Thumb drive and partitions
B. Building file system in a partition
C. Basic design of FAT file system
   - Metadata vs. Real data
D. Deleted file recovery mechanisms
   - Metadata-based
   - File carving
E. A hands-on case study

A hands-on case study follows

- This demonstrates a few items
  - How to create FAT file system in a thumb drive on a Windows computer
  - How to attach the file system on a Linux virtual machine
  - What happens in the file system when we create a new file
  - What happens when the file is deleted
  - How metadata-based mechanism can recover the deleted file

Creating a partition and a file system

- A thumb drive is attached on a Windows computer.
- The figure shows Disk Management tool of Windows being used
  - Disk 0 is the hard disk whereas Disk 1 is the thumb drive.
  - A 4GB partition is created in the thumb drive
  - Then, we format that 4GB partition to build an instance of FAT32 system.
  - This partition is also labeled as FORENSICS (D) drive.
  - The other part (53.28 GB) of the thumb drive is unallocated.



Volume	Layout	Type	File System	Status	Capacity	Free Sp...	% Free
(Disk 0 partition 1)	Simple	Basic		Healthy (E...	500 MB	500 MB	100 %
FORENSICS (D:)	Simple	Basic	FAT32	Healthy (P...	3.99 GB	3.99 GB	100 %
Windows (C:)	Simple	Basic	NTFS (BitLo...	Healthy (B...	476.33 GB	154.21 GB	32 %

Disk 0
Basic
476.81 GB
Online

500 MB
Healthy (EFI System Partition)

Windows (C:)
476.33 GB NTFS (BitLocker Encrypted)
Healthy (Boot, Page File, Crash Dump, Basic Data Partition)

Disk 1
Removable
57.28 GB
Online

FORENSICS (D:)
4.00 GB FAT32
Healthy (Primary Partition)

53.28 GB
Unallocated

Kali VM

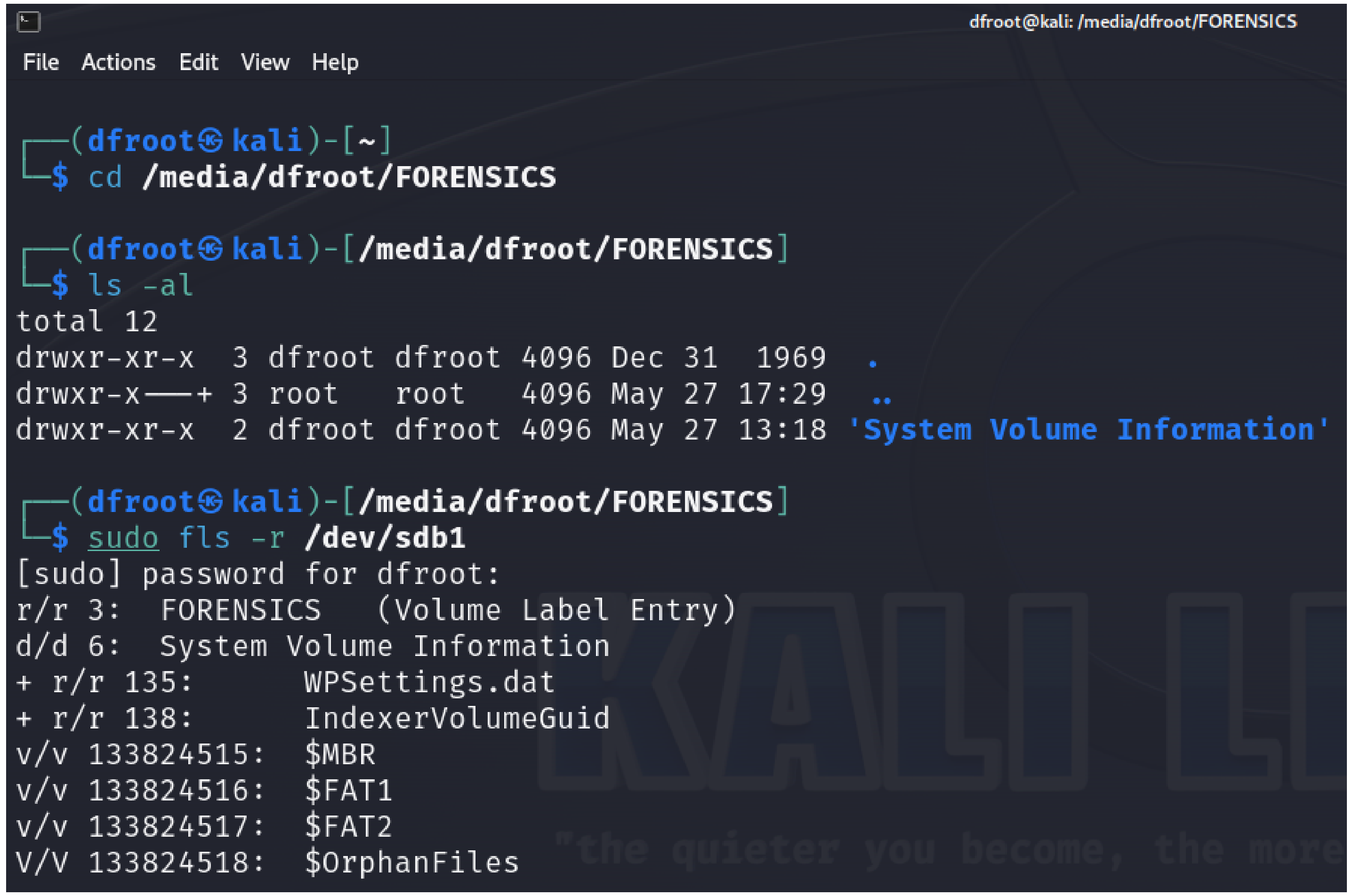
- Install the Virtualbox software on your Windows computer
- Download Kali VM from the official website
  - How to run Kali VM on virtualbox and more details are in Appendix
  - Create a sudo user (e.g., dfroot) in the kali VM
- Connect the thumb drive to the VM
  - Using the "Device" button on the VM
  - The lsblk command identifies the thumb drive
    - As an example, the thumb drive is identified as /dev/sdb whereas the partition is identified as /dev/sdb1

Kali VM (contd.)

- Mount the FORENSICS partition (that we created before)
  - Use this command to mount: `udisksctl  mount -b /dev/sdb1`
  - As an example, the default mountpoint can be /media/dfroot/FORENSICS
- Kali VM has the TSK toolkit already installed
  - TSK is the inner engine of Autopsy; TSK comes with kali VM by default.
  - TSK is a set of commands that run on command line
  - We use multiple commands in TSK, such as fsstat, fls, istat, icat, and more

Checking the content of the thumb drive

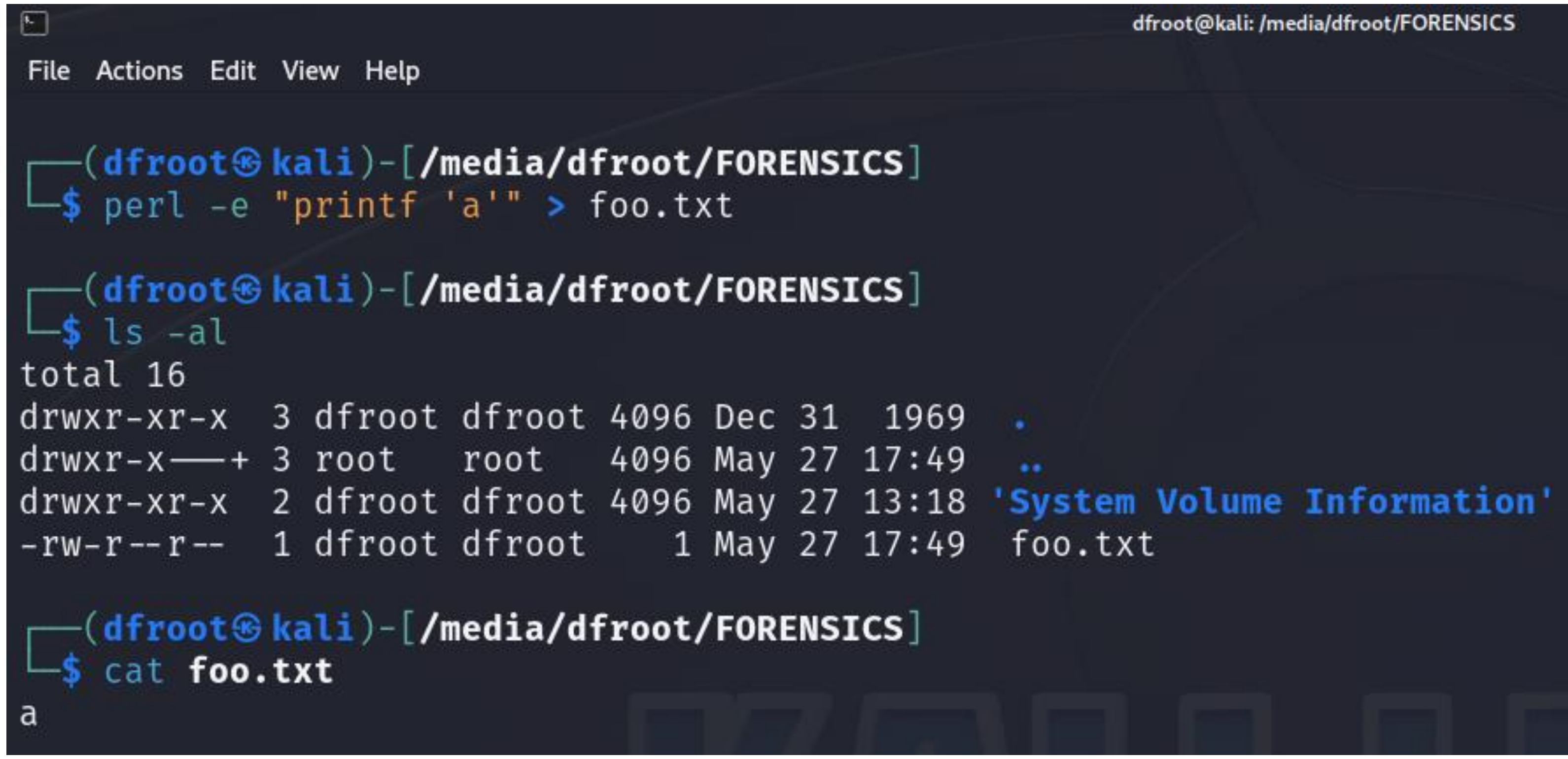
- cd mountpoint
- ls command shows the content of the FORENSICS drive.
  - The disk management tool created some directory and files by default
- fls command shows the content of the FORENSICS drive with more details.
  - d/d represents a directory
  - r/r represents a file
  - v/v represents a virtual file (only for TSK itself)
- We see there is no file named foo.txt that we plan to create next.


dfroot@kali: /media/dfroot/FORENSICS
File Actions Edit View Help

```
┌──(dfroot㉿kali)-[~]
└─$ cd /media/dfroot/FORENSICS

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ ls -al
total 12
drwxr-xr-x  3 dfroot dfroot 4096 Dec 31  1969  .
drwxr-x---+ 3 root   root   4096 May 27 17:29  ..
drwxr-xr-x  2 dfroot dfroot 4096 May 27 13:18 'System Volume Information'

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo fls -r /dev/sdb1
[sudo] password for dfroot:
r/r 3:  FORENSICS   (Volume Label Entry)
d/d 6:  System Volume Information
+ r/r 135:      WPSettings.dat
+ r/r 138:      IndexerVolumeGuid
v/v 133824515:  $MBR
v/v 133824516:  $FAT1
v/v 133824517:  $FAT2
V/V 133824518:  $OrphanFiles
```

Creating a new file in the FORENSICS Drive

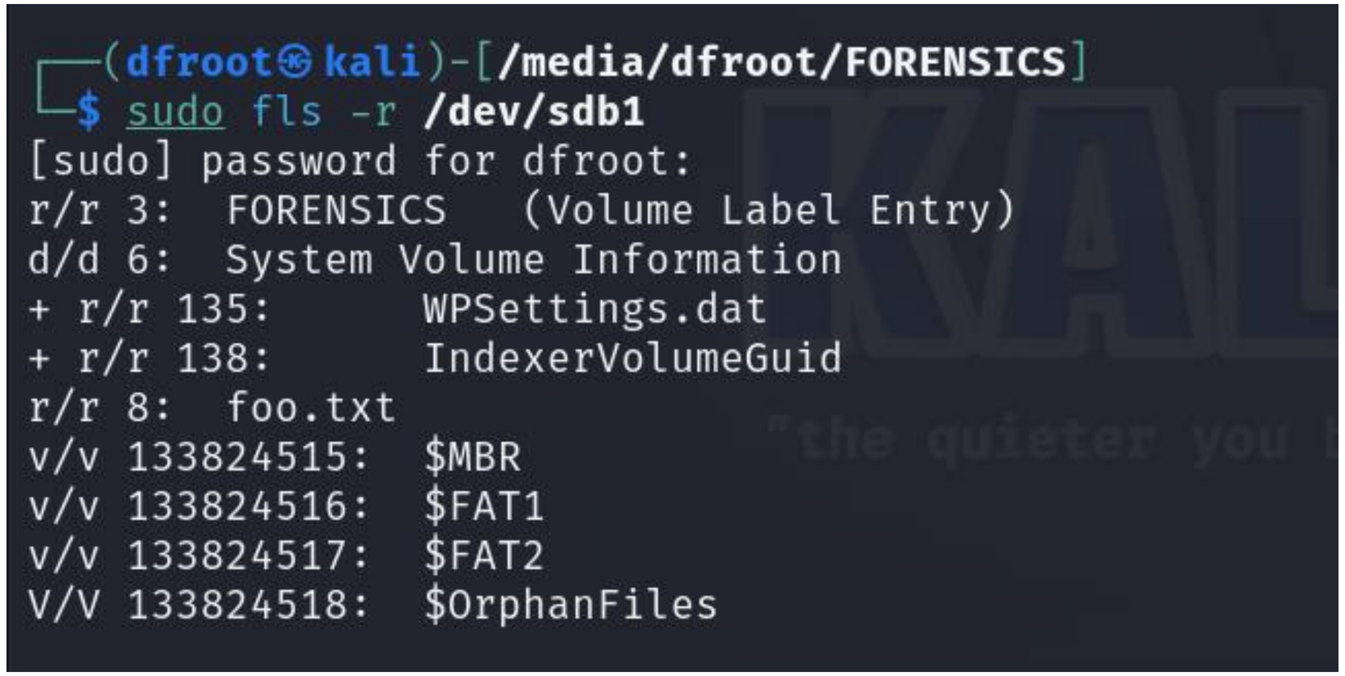
- The perl command just prints one character ‘a’ to a new file, foo.txt
- The ls command lists the new file.
  - The size of foo.txt is just 1 byte, which is just one character (‘a’).
  - However, we note that the name ‘foo.txt’ has 7 characters
  - This shows that name of a file is not even part of real data; actually, the name is part of metadata.
- The cat command confirms that foo.txt’s sole content is one ‘a’.


dfroot@kali: /media/dfroot/FORENSICS
File Actions Edit View Help

```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ perl -e "printf 'a'" > foo.txt

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ ls -al
total 16
drwxr-xr-x  3 dfroot dfroot 4096 Dec 31  1969  .
drwxr-x---+ 3 root   root   4096 May 27 17:49  ..
drwxr-xr-x  2 dfroot dfroot 4096 May 27 13:18 'System Volume Information'
-rw-r--r--  1 dfroot dfroot    1 May 27 17:49  foo.txt

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ cat foo.txt
a
```

Rechecking the content of the FORENSICS drive

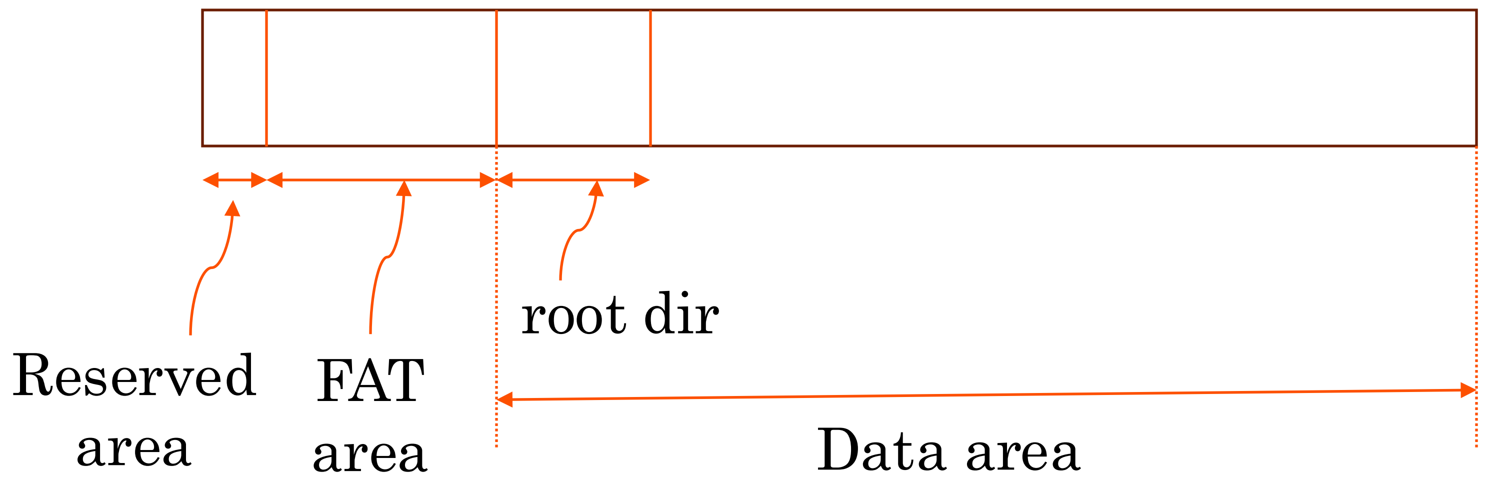
- The fls command does identify the new file, foo.txt
- The TSK toolkit (which fls is part of) assigns one unique number (called inode number) to each file or folder.
    - The inode number of foo.txt is 8
    - d/d represents a directory whereas r/r represents a file.
    - v/v represents a virtual file, which TSK creates for tracking purpose.

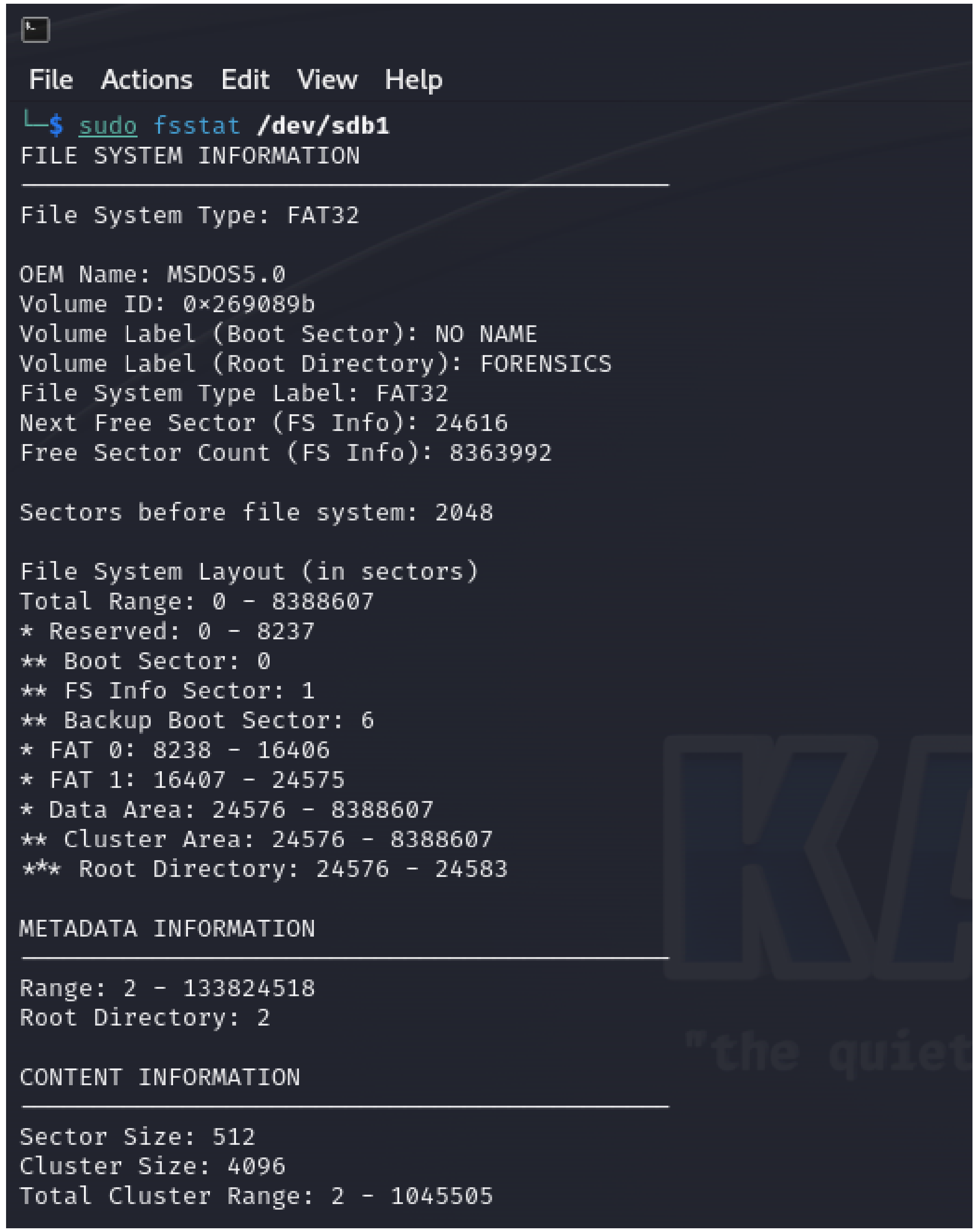


```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo fls -r /dev/sdb1
[sudo] password for dfroot:
r/r 3:  FORENSICS   (Volume Label Entry)
d/d 6:  System Volume Information
+ r/r 135:      WPSettings.dat
+ r/r 138:      IndexerVolumeGuid
r/r 8:  foo.txt
v/v 133824515:  $MBR
v/v 133824516:  $FAT1
v/v 133824517:  $FAT2
V/V 133824518:  $OrphanFiles
```

Checking the file system information

- The fsstat command shows the layout of the FAT32 file system
- Three main areas in the layout
  - Reserved area, which starts with boot sector.
  - FAT area, which contains two copies of FAT table.
  - Data area, which starts with root directory.
- One cluster contains 8 sectors.
  - Sector range: 0 to 8388607
  - Cluster range: 2 to 1045505

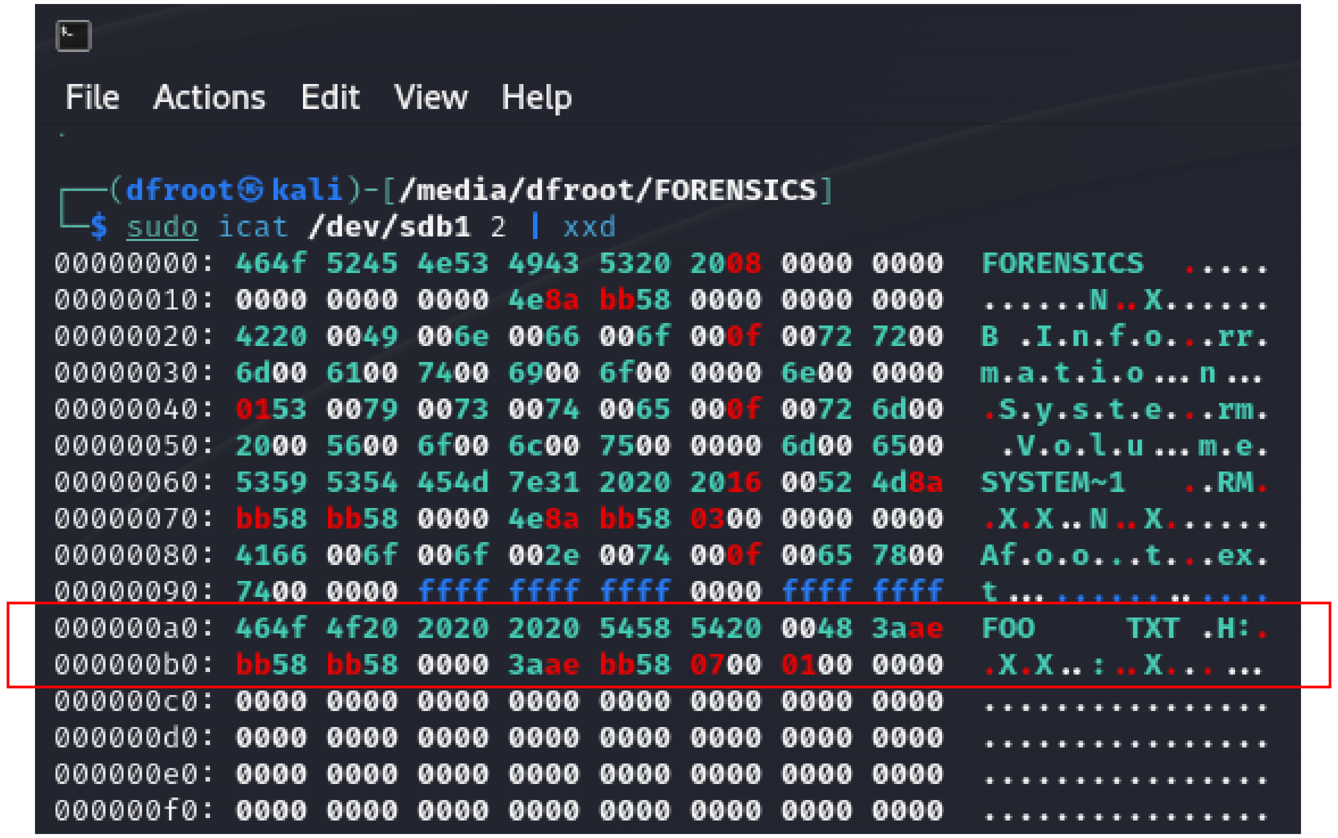




```
└─$ sudo fsstat /dev/sdb1
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: FAT32

OEM Name: MSDOS5.0
Volume ID: 0x269089b
Volume Label (Boot Sector): NO NAME
Volume Label (Root Directory): FORENSICS
File System Type Label: FAT32
Next Free Sector (FS Info): 24616
Free Sector Count (FS Info): 8363992

Sectors before file system: 2048

File System Layout (in sectors)
Total Range: 0 - 8388607
* Reserved: 0 - 8237
** Boot Sector: 0
** FS Info Sector: 1
** Backup Boot Sector: 6
* FAT 0: 8238 - 16406
* FAT 1: 16407 - 24575
* Data Area: 24576 - 8388607
** Cluster Area: 24576 - 8388607
*** Root Directory: 24576 - 24583

METADATA INFORMATION
--------------------------------------------
Range: 2 - 133824518
Root Directory: 2

CONTENT INFORMATION
--------------------------------------------
Sector Size: 512
Cluster Size: 4096
Total Cluster Range: 2 - 1045505
```

Checking the root directory of FORENSICS drive

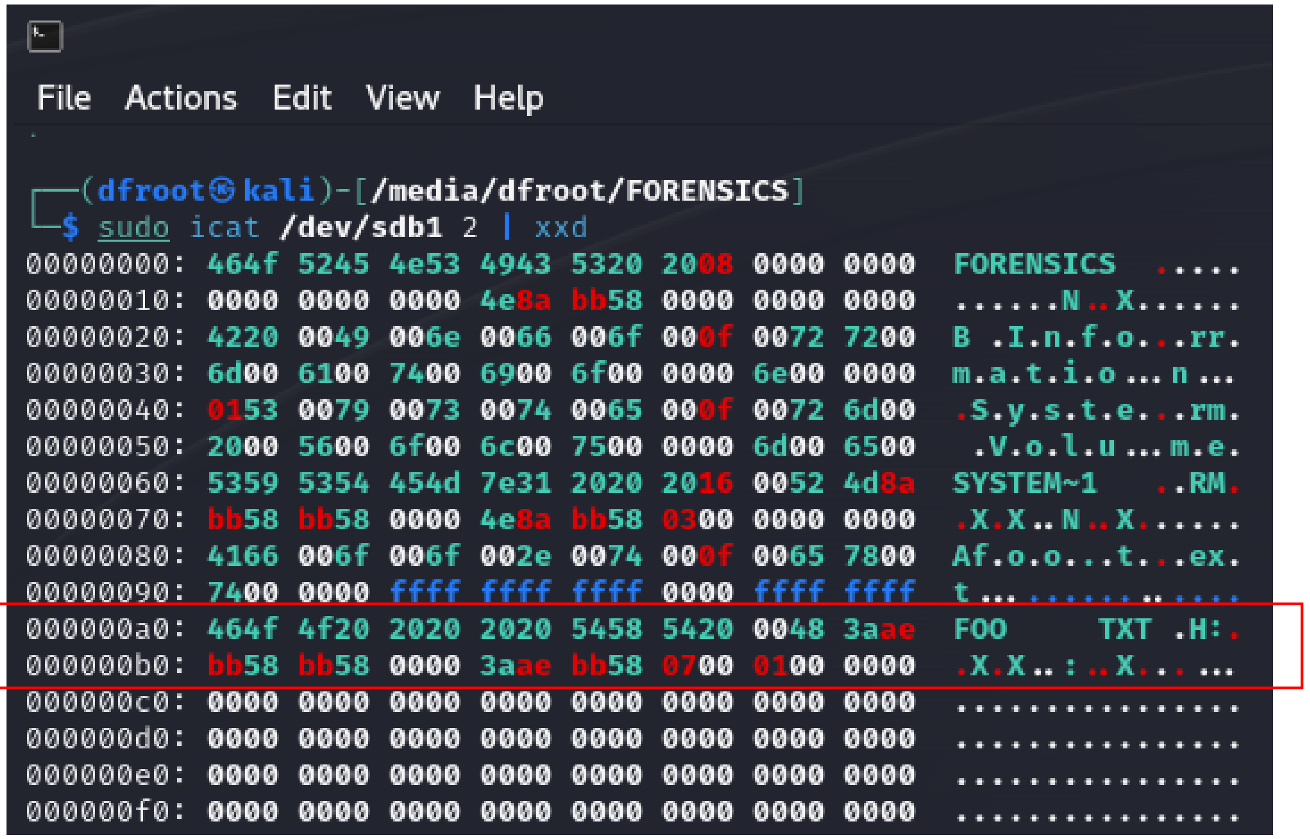
- Let’s use icat command (part of TSK toolkit) to check the content of the root dir.
  - Using the inode number of any file or folder, icat command can show the content
  - Note that the inode number of root directory is always 2
  - The raw output of icat is piped to another tool named xxd; the pipe symbol is |
- xxd visualizes raw content in a friendly way
  - It shows 16 bytes of content in each row
  - The output has 3 columns: left column is the byte index, middle column shows the bytes in hexadecimal, and right column shows the corresponding English text if any.
  - As an example, we see 0th byte is 46 (in hex) that represents ‘F’ (in ASCII code), and 1st byte is 4f (in hex) that represents ‘R’ (in ASCII code), and more.
- Let’s focus on the last two rows (32 bytes), which is the directory entry for foo.txt.



```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo icat /dev/sdb1 2 | xxd
00000000: 464f 5245 4e53 4943 5320 2008 0000 0000  FORENSICS  .....
00000010: 0000 0000 0000 4e8a bb58 0000 0000 0000  ......N..X......
00000020: 4220 0049 006e 0066 006f 000f 0072 7200  B .I.n.f.o...rr.
00000030: 6d00 6100 7400 6900 6f00 0000 6e00 0000  m.a.t.i.o...n...
00000040: 0153 0079 0073 0074 0065 000f 0072 6d00  .S.y.s.t.e...rm.
00000050: 2000 5600 6f00 6c00 7500 0000 6d00 6500   .V.o.l.u...m.e.
00000060: 5359 5354 454d 7e31 2020 2016 0052 4d8a  SYSTEM~1   ..RM.
00000070: bb58 bb58 0000 4e8a bb58 0300 0000 0000  .X.X..N..X......
00000080: 4166 006f 006f 002e 0074 000f 0065 7800  Af.o.o...t...ex.
00000090: 7400 0000 ffff ffff ffff 0000 ffff ffff  t...............
000000a0: 464f 4f20 2020 2020 5458 5420 0048 3aae  FOO     TXT .H:.
000000b0: bb58 bb58 0000 3aae bb58 0700 0100 0000  .X.X..:..X......
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Inspecting the directory entry of foo.txt

- Let’s identify a few items in the directory entry of foo.txt, which spans row a0 and b0
- Byte 0-10 represents the file name
  - We see: FOO   TXT
  - Byte 0 is 46 (hex) which represents ‘F’, and so on
- Byte 26-27 represents the starting cluster’s index, which hosts the file data
  - We see: 0700 (little endian); this means cluster 7 hosts the data of foo.txt
- Byte 28-31 represents the file size
  - We see: 0100 0000 (little endian); this means the file size is 1 byte

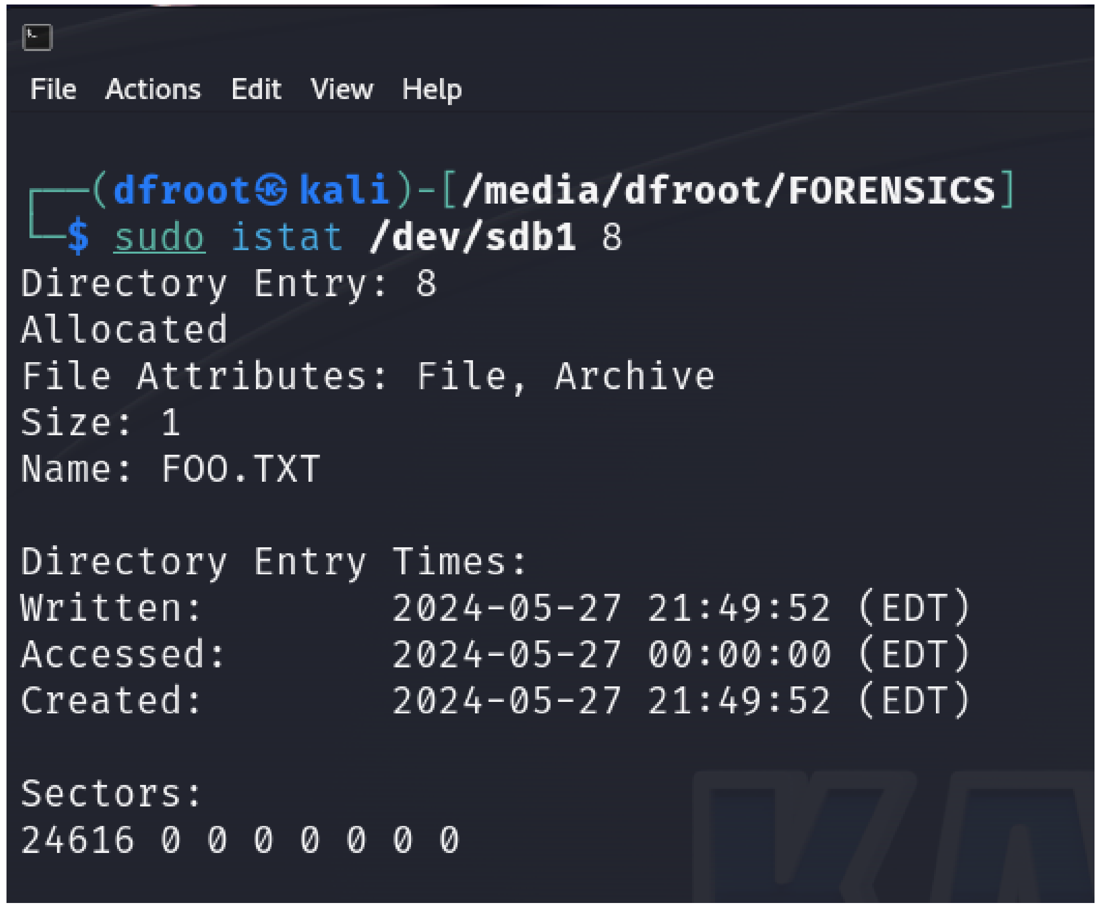


```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo icat /dev/sdb1 2 | xxd
00000000: 464f 5245 4e53 4943 5320 2008 0000 0000  FORENSICS  .....
00000010: 0000 0000 0000 4e8a bb58 0000 0000 0000  ......N..X......
00000020: 4220 0049 006e 0066 006f 000f 0072 7200  B .I.n.f.o...rr.
00000030: 6d00 6100 7400 6900 6f00 0000 6e00 0000  m.a.t.i.o...n...
00000040: 0153 0079 0073 0074 0065 000f 0072 6d00  .S.y.s.t.e...rm.
00000050: 2000 5600 6f00 6c00 7500 0000 6d00 6500   .V.o.l.u...m.e.
00000060: 5359 5354 454d 7e31 2020 2016 0052 4d8a  SYSTEM~1   ..RM.
00000070: bb58 bb58 0000 4e8a bb58 0300 0000 0000  .X.X..N..X......
00000080: 4166 006f 006f 002e 0074 000f 0065 7800  Af.o.o...t...ex.
00000090: 7400 0000 ffff ffff ffff 0000 ffff ffff  t...............
000000a0: 464f 4f20 2020 2020 5458 5420 0048 3aae  FOO     TXT .H:.
000000b0: bb58 bb58 0000 3aae bb58 0700 0100 0000  .X.X..:..X......
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

Note. The previous 32 bytes (row 80 and row 90) represent an additional directory entry for the same file, which co
richer encoding of the file name. We will mostly ignore this.

Inspecting stats of foo.txt

- We use istat command (part of TSK)
  - It takes the partition name (/dev/sdb1) and inode number (8) of the file as parameters
- It reports the size of the file as 1 byte
  - which is consistent with what we saw before.
- Sector 24616 hosts the content of the file
  - We knew cluster 7 hosts the content of the file
  - The index of the first sector in Cluster 7 is indeed 24616

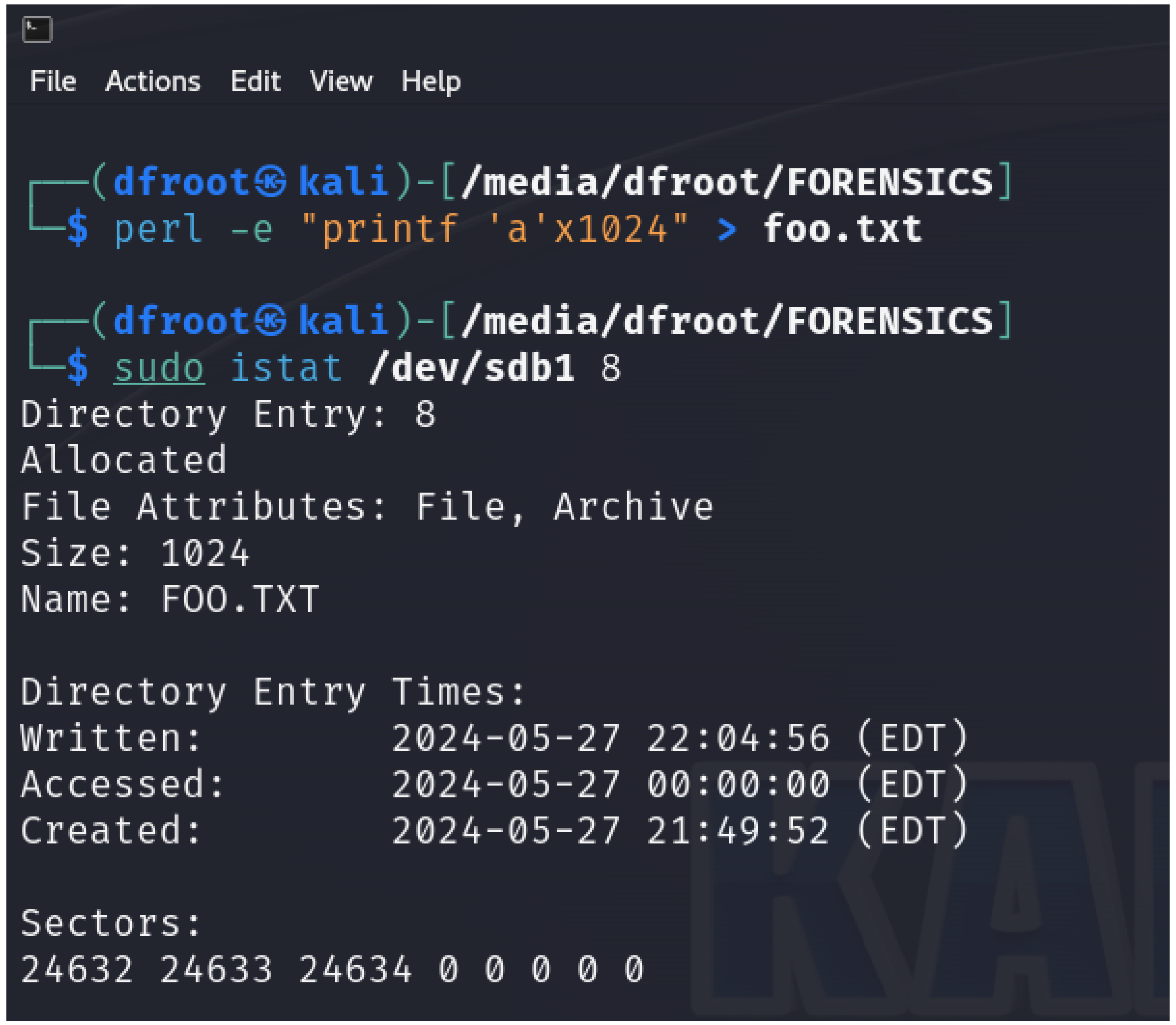


```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo istat /dev/sdb1 8
Directory Entry: 8
Allocated
File Attributes: File, Archive
Size: 1
Name: FOO.TXT

Directory Entry Times:
Written:        2024-05-27 21:49:52 (EDT)
Accessed:       2024-05-27 00:00:00 (EDT)
Created:        2024-05-27 21:49:52 (EDT)

Sectors:
24616 0 0 0 0 0 0 0
```

fsstat told us before that cluster 2 (root directory) starts at Sector 24576.
So, cluster 7 starts at 24576 + (7-2)*8 = 24616th sector

What happens if we make foo.txt longer

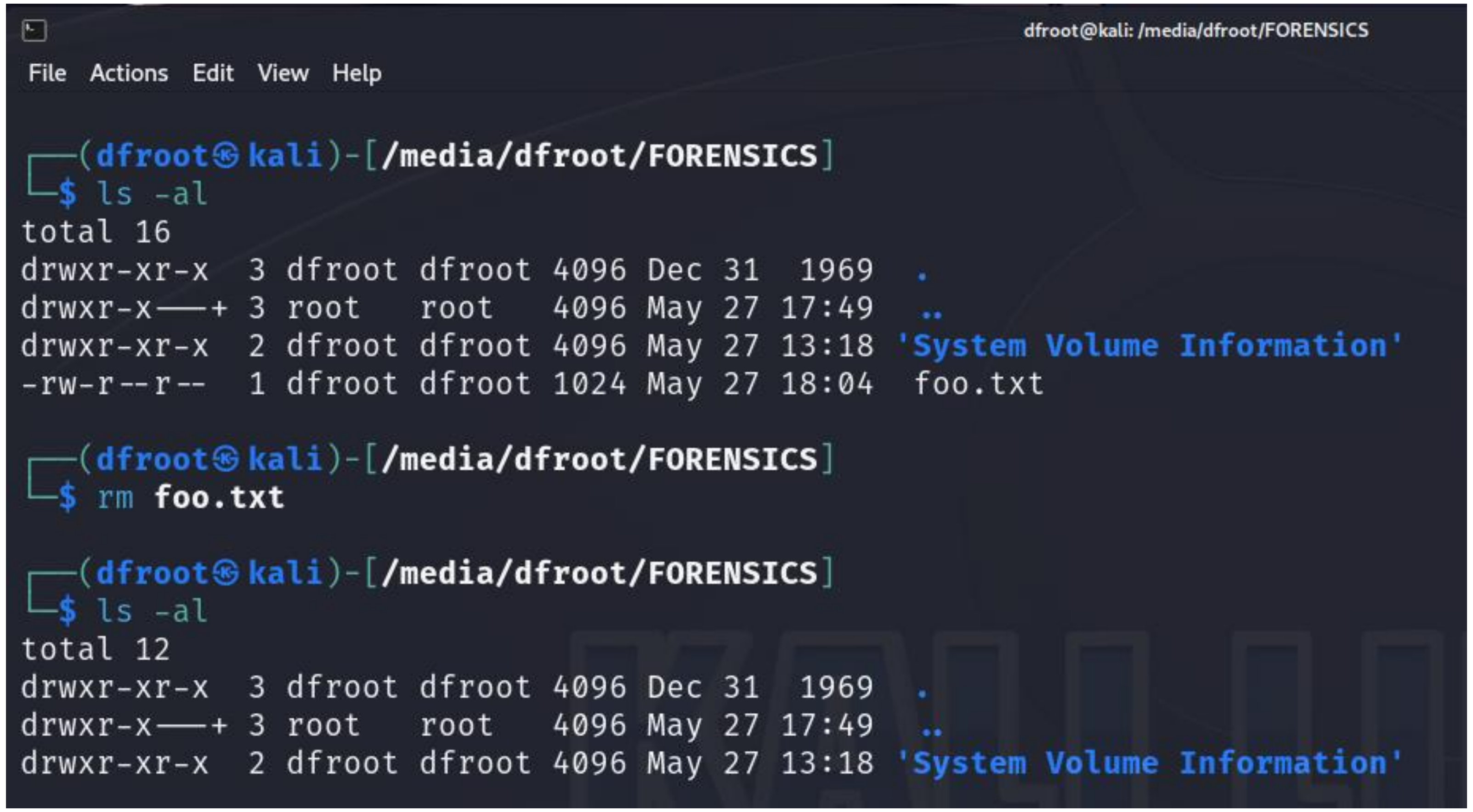
- Let’s inject 1024 characters (all ‘a’s) in foo.txt
  - We use the same perl command
- Use istat command to check the current stats of foo.txt
  - Now it takes more than one Sectors to host the content of foo.txt
- Note that one sector, which is 512 byte, can host only so many characters



```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ perl -e "printf 'a'x1024" > foo.txt

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo istat /dev/sdb1 8
Directory Entry: 8
Allocated
File Attributes: File, Archive
Size: 1024
Name: FOO.TXT

Directory Entry Times:
Written:        2024-05-27 22:04:56 (EDT)
Accessed:       2024-05-27 00:00:00 (EDT)
Created:        2024-05-27 21:49:52 (EDT)

Sectors:
24632 24633 24634 0 0 0 0 0
```

Now let's delete foo.txt

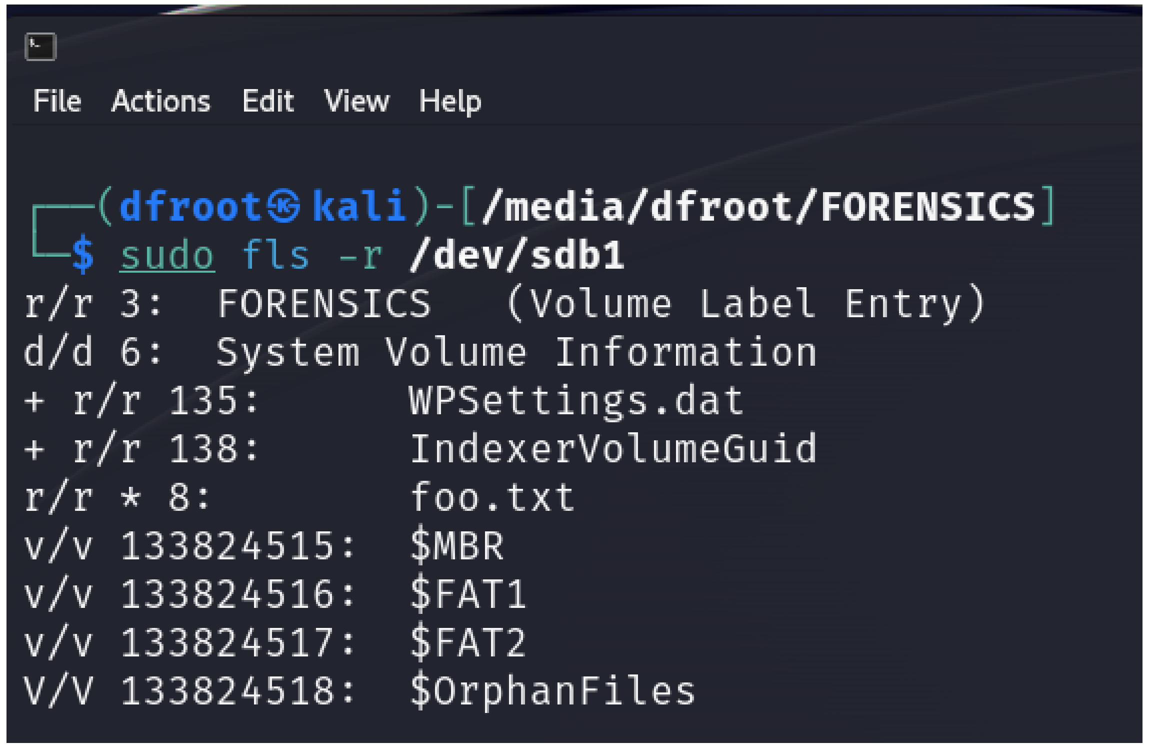
- We use the rm command to delete foo.txt
- After that, ls command no longer identifies foo.txt


dfroot@kali: /media/dfroot/FORENSICS
File Actions Edit View Help

```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ ls -al
total 16
drwxr-xr-x  3 dfroot dfroot 4096 Dec 31  1969 .
drwxr-x---+ 3 root   root   4096 May 27 17:49 ..
drwxr-xr-x  2 dfroot dfroot 4096 May 27 13:18 'System Volume Information'
-rw-r--r--  1 dfroot dfroot 1024 May 27 18:04  foo.txt

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ rm foo.txt

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ ls -al
total 12
drwxr-xr-x  3 dfroot dfroot 4096 Dec 31  1969 .
drwxr-x---+ 3 root   root   4096 May 27 17:49 ..
drwxr-xr-x  2 dfroot dfroot 4096 May 27 13:18 'System Volume Information'
```

Can fls identify deleted foo.txt?

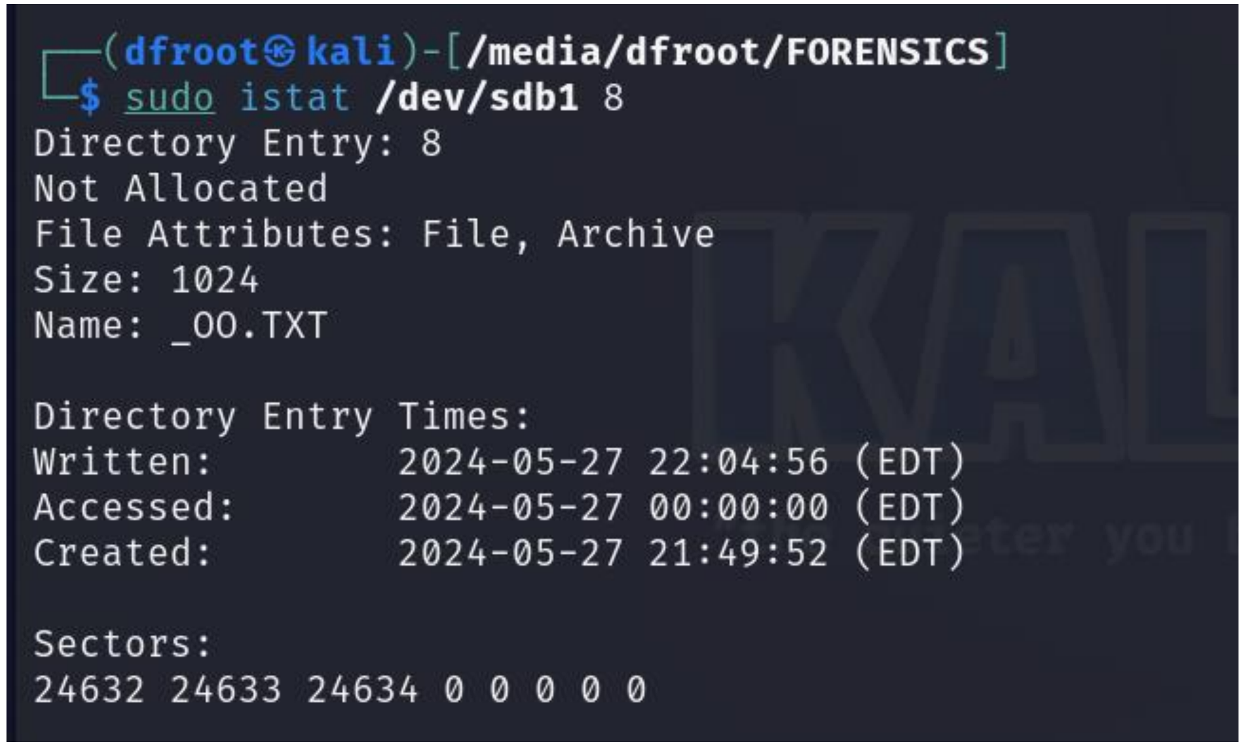
- Yes, it can.
  - We run fls on /dev/sdb1 partition
  - fls still associates the same inode number (8) with foo.txt
  - * indicates that the file is deleted



```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo fls -r /dev/sdb1
r/r 3:   FORENSICS   (Volume Label Entry)
d/d 6:   System Volume Information
+ r/r 135:      WPSettings.dat
+ r/r 138:      IndexerVolumeGuid
r/r * 8:        foo.txt
v/v 133824515:  $MBR
v/v 133824516:  $FAT1
v/v 133824517:  $FAT2
V/V 133824518:  $OrphanFiles
```

What information does istat show at this point?

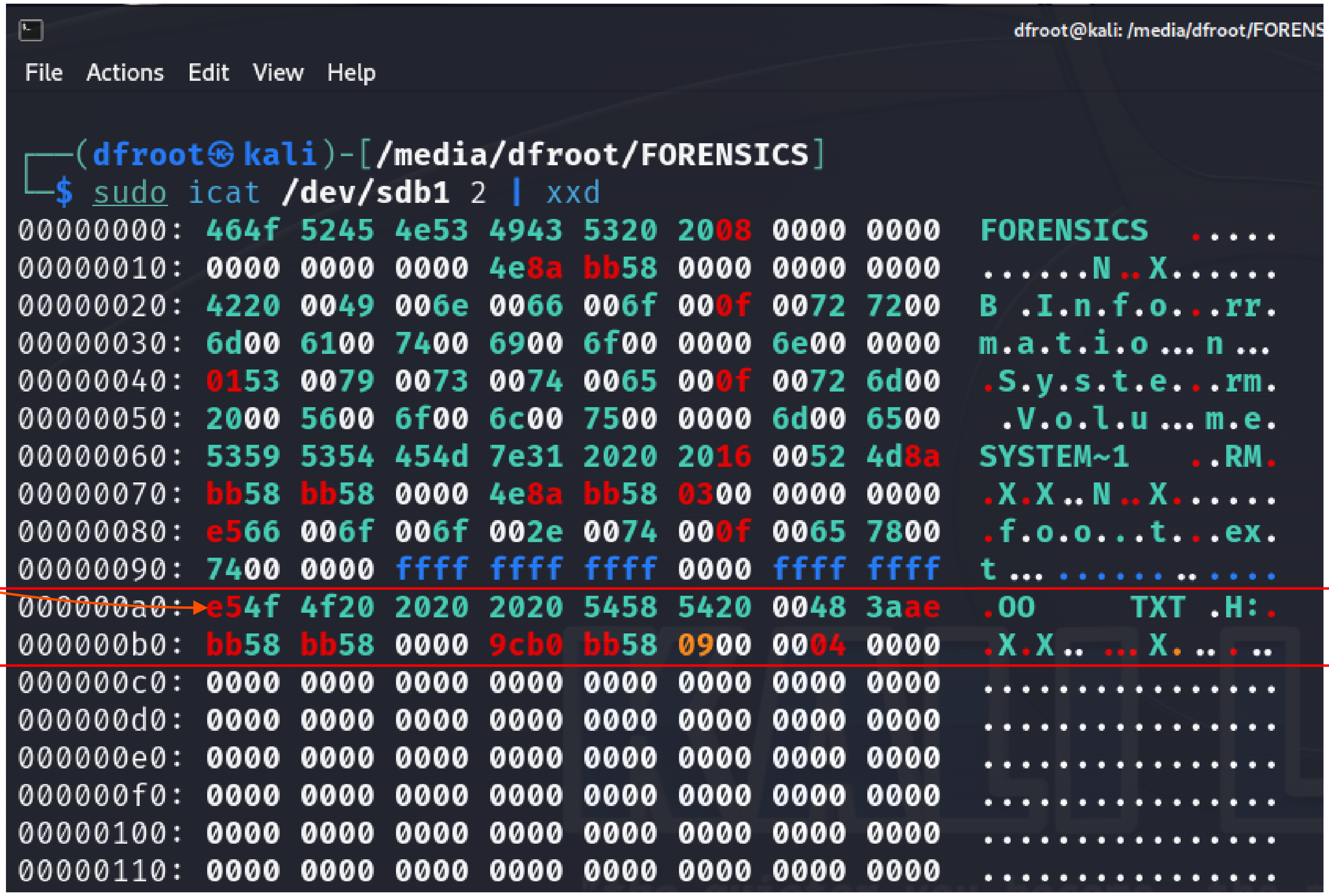
- We run istat command on the same partition (/dev/sdb1) for the inode number (8) of foo.txt
- The istat output on foo.txt is very similar to what it was before the deletion step
- The main difference is the first character ('F') of the filename is now replaced with an underscore ('_').
- We also check that the content of foo.txt still remains; the corresponding Sectors (e.g. Sector 24632) still host 1024 'a's.



```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo istat /dev/sdb1 8
Directory Entry: 8
Not Allocated
File Attributes: File, Archive
Size: 1024
Name: _OO.TXT

Directory Entry Times:
Written:        2024-05-27 22:04:56 (EDT)
Accessed:       2024-05-27 00:00:00 (EDT)
Created:        2024-05-27 21:49:52 (EDT)

Sectors:
24632 24633 24634 0 0 0 0 0
```

Inspecting the directory entry of deleted foo.txt

- Byte 0-10 represents the file name
  - We see: .OO   TXT
  - Byte 0 is e5 (hex) which was 46 (hex)
- Byte 26-27 represents the starting cluster's index, which hosts the file data
  - We see: 0900 (little endian); this means the data of foo.txt starts from cluster 9
- Byte 28-31 represents the file size
  - We see: 0004 0000 (little endian); this means the file size is 1024 bytes

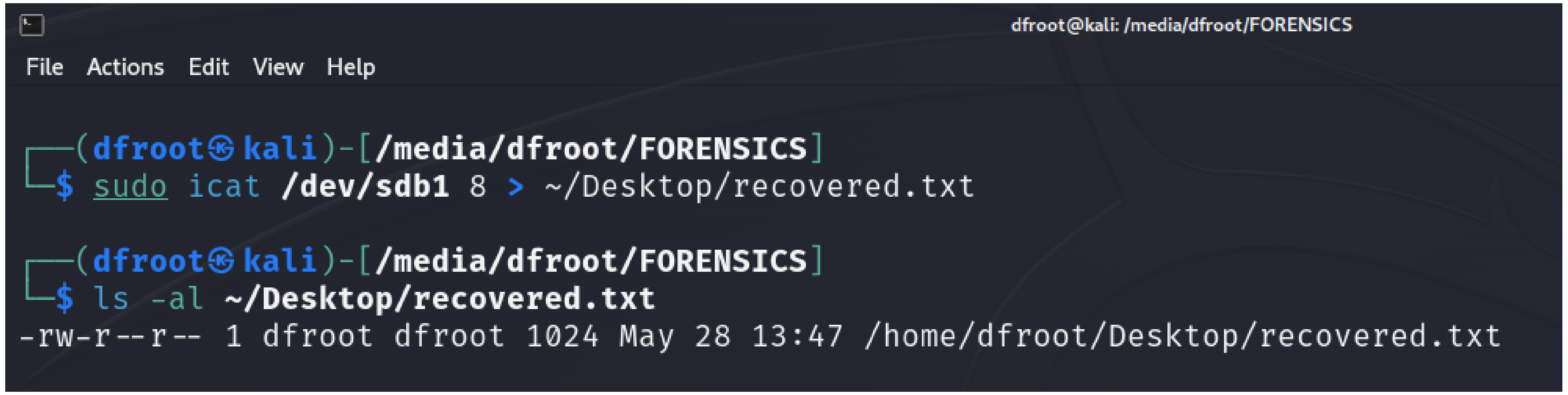

dfroot@kali: /media/dfroot/FORENS
File Actions Edit View Help

```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo icat /dev/sdb1 2 | xxd
00000000: 464f 5245 4e53 4943 5320 2008 0000 0000  FORENSICS  .....
00000010: 0000 0000 0000 4e8a bb58 0000 0000 0000  ......N..X......
00000020: 4220 0049 006e 0066 006f 000f 0072 7200  B .I.n.f.o...rr.
00000030: 6d00 6100 7400 6900 6f00 0000 6e00 0000  m.a.t.i.o...n...
00000040: 0153 0079 0073 0074 0065 000f 0072 6d00  .S.y.s.t.e...rm.
00000050: 2000 5600 6f00 6c00 7500 0000 6d00 6500   .V.o.l.u...m.e.
00000060: 5359 5354 454d 7e31 2020 2016 0052 4d8a  SYSTEM~1   ..RM.
00000070: bb58 bb58 0000 4e8a bb58 0300 0000 0000  .X.X..N..X......
00000080: e566 006f 006f 002e 0074 000f 0065 7800  .f.o.o...t...ex.
00000090: 7400 0000 ffff ffff ffff 0000 ffff ffff  t...............
000000a0: e54f 4f20 2020 2020 5458 5420 0048 3aae  .OO     TXT .H:.
000000b0: bb58 bb58 0000 9cb0 bb58 0900 0004 0000  .X.X.....X......
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

This means the directory entry of the file still exists (in the parent directory) even after the file is deleted. The TSK toolkit (e.g., icat command) can use this metadata (aka. directory entry) to recover the deleted file.

Recovering deleted foo.txt

- Recall that the inode number of foo.txt is 8
- We use icat command to retrieve foo.txt and output is saved as recovered.txt
- ls command shows that size of recovered.txt is 1024 bytes, which is same as foo.txt
- We can also open recovered.txt on an editor to check that it has the same content as foo.txt.


dfroot@kali: /media/dfroot/FORENSICS
File Actions Edit View Help

```
┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ sudo icat /dev/sdb1 8 > ~/Desktop/recovered.txt

┌──(dfroot㉿kali)-[/media/dfroot/FORENSICS]
└─$ ls -al ~/Desktop/recovered.txt
-rw-r--r-- 1 dfroot dfroot 1024 May 28 13:47 /home/dfroot/Desktop/recovered.txt
```

This shows that metadata-based deleted file recovery was successful.

Summary

- A file has metadata and real data
- After the file is deleted metadata and/or real data may remain
- We saw an example of how to use metadata to recover a deleted file
- Sometimes metadata may not be available, e.g., in following cases
  - The partition went through the “quick formatting” process
  - Metadata got overwritten due to new file creatation
- Then, we can use another recovery mechanism known as file carving
  - This mechanism works only if the file data has a header and footer signature
  - As no metadata is required, it is agnostic to the file system type. Thus, a file carving technique works the same for FAT file system and another file system.

Any question?

THANK YOU!

BGSU. A PUBLIC UNIVERSITY FOR THE PUBLIC GOOD.