Operating Systems

Associate Prof. Yongkun Li
中科大-计算机学院 副教授
http://staff.ustc.edu.cn/~ykli

Ch10, part 1
Details of FAT32

Story so far…

What are stored on disk

File: content + attributes
Directory: Directory file

How to access them?

File operations: `open(), read(), write()`
Directory lookup: Directory traversal

How are the files stored on disk?

File system layout

Contiguous allocation
linked-list allocation (FAT*)
index-node allocation (EXT*)

Topics in Ch10

- Case study

Details of FAT32

File attributes and directory entries, file operations

Details of Ext2/3/4

Detailed layout, detailed inode structure (file attributes), FS operations…

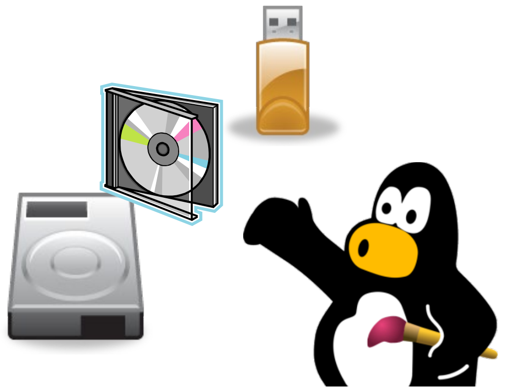
Cutting-edge systems

Key-value systems

Details of FAT32

- **Introduction**
- Directory and File Attributes
- File Operations
  - Read files
  - Write files
  - Delete files
  - Recover deleted files

Microsoft Extensible Firmware Initiative FAT32 File System Specification (FAT: General Overview of On-Disk Format), Version 1.03, December 6, 2000, hardware white papers @ Microsoft Corporation.

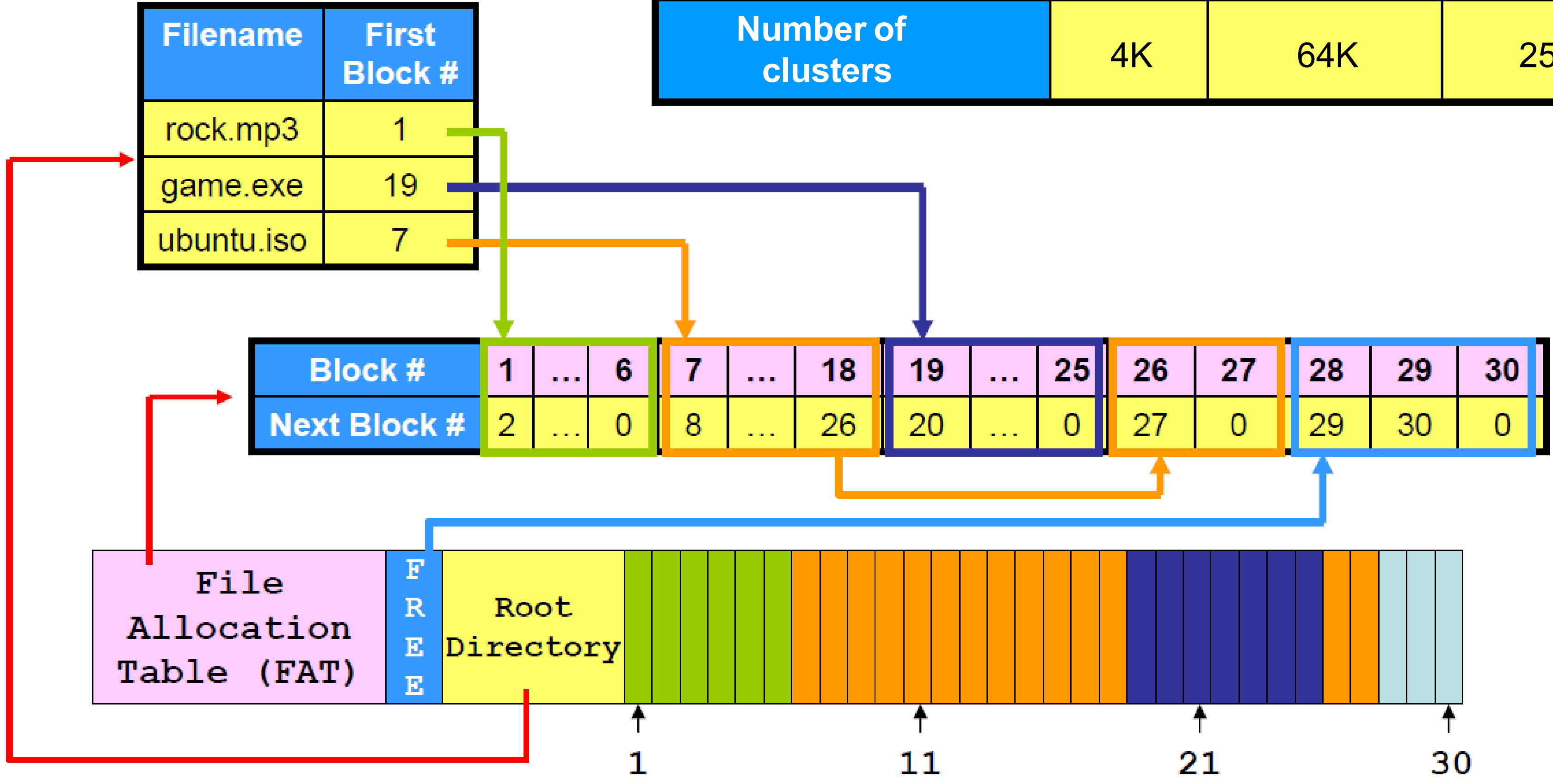


Recall on FAT allocation

- The layout

A block is named a **cluster**.

File System	FAT12	FAT16	FAT32
Cluster addr length	12 bits	16 bits	32 bits (28?)
Number of clusters	4K	64K	256M



Trivia

- Cluster Size:

When a sector is > 512B …

512B	1KB	2KB	4KB	8KB	16KB	32KB	64KB	128KB	256KB

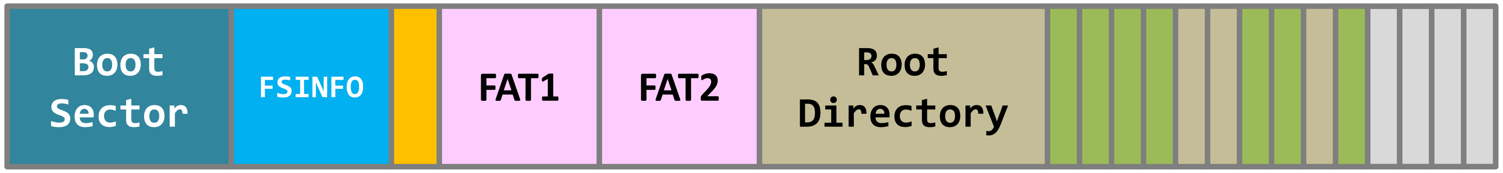
  – Try typing “help format” in the command prompt in Windows.

- Calculating the maximum partition size
  – with the cluster size = 32KB…

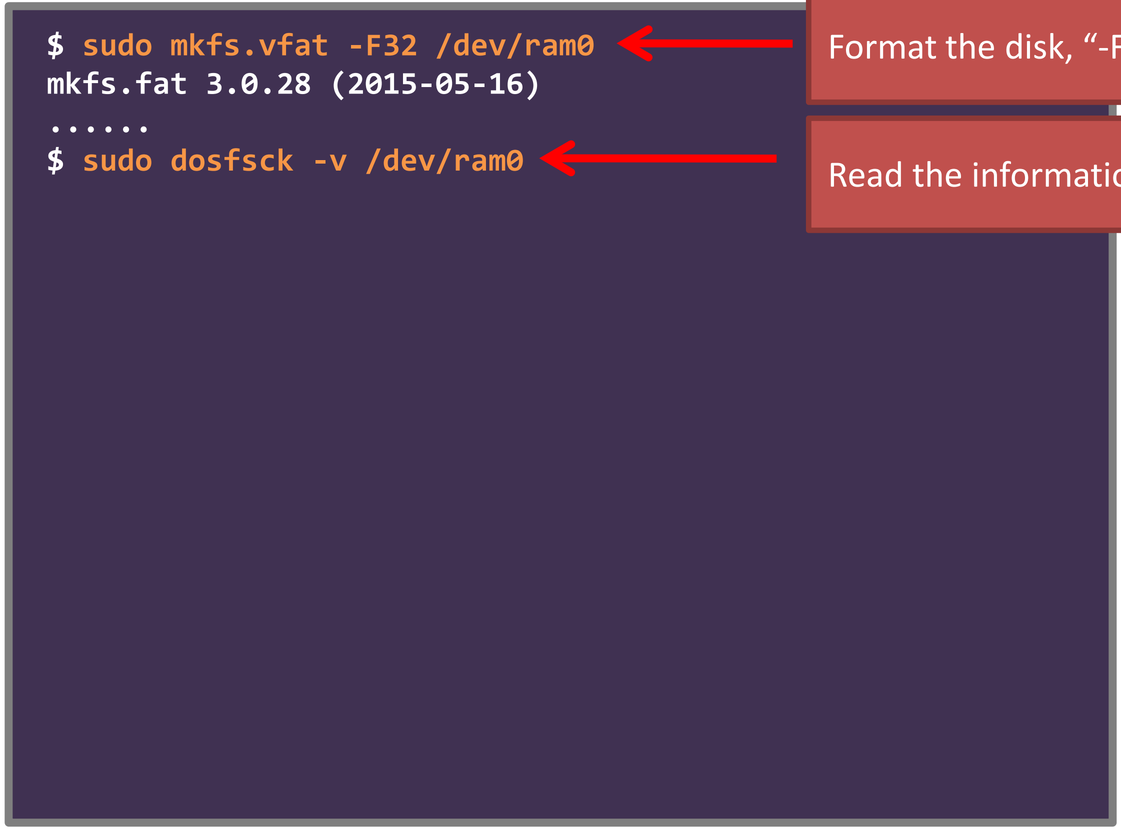
$$(32 \times 2^{10}) \times 2^{28} = 2^{43}\ (8TB)$$

Typical layout of a FAT32 partition

	Propose	Size
Boot sector	Store FS-specific parameters	1 sector, 512 bytes
FSINFO	Free-space management	1 sector, 512 bytes
Reserved sectors	**Don't ask me, ask Micro$oft!**	Variable, can be changed during format.
FAT (2 pieces)	A **robust design**: if "**FAT 1**" is corrupted or containing bad sectors, then "**FAT 2**" can act as a backup.	Variable, depends on disk size and cluster size.
Root directory	Start of the directory tree.	At least one cluster, depend on the number of director entries.



Typical layout of a FAT32 partition



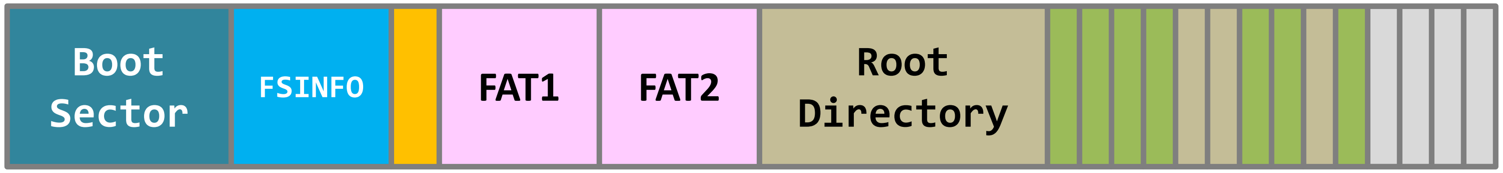
```
$ sudo mkfs.vfat -F32 /dev/ram0
mkfs.fat 3.0.28 (2015-05-16)
......
$ sudo dosfsck -v /dev/ram0
```

Format the disk, "-F32" means FAT32.

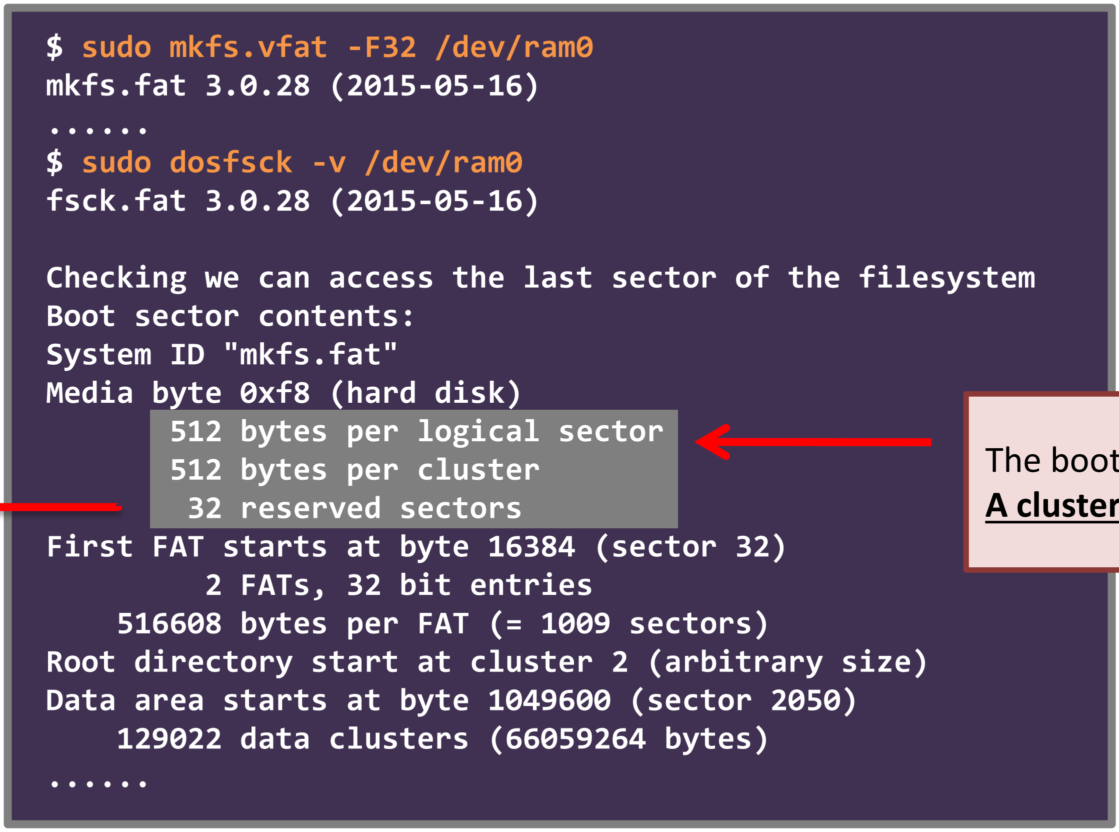
Read the information stored in the boot sector.

Running "**dosfsck**", *DOS file system check*, on a FAT32 FS.

This program reads details from the **Boot Sector**.



Typical layout of a FAT32 partition

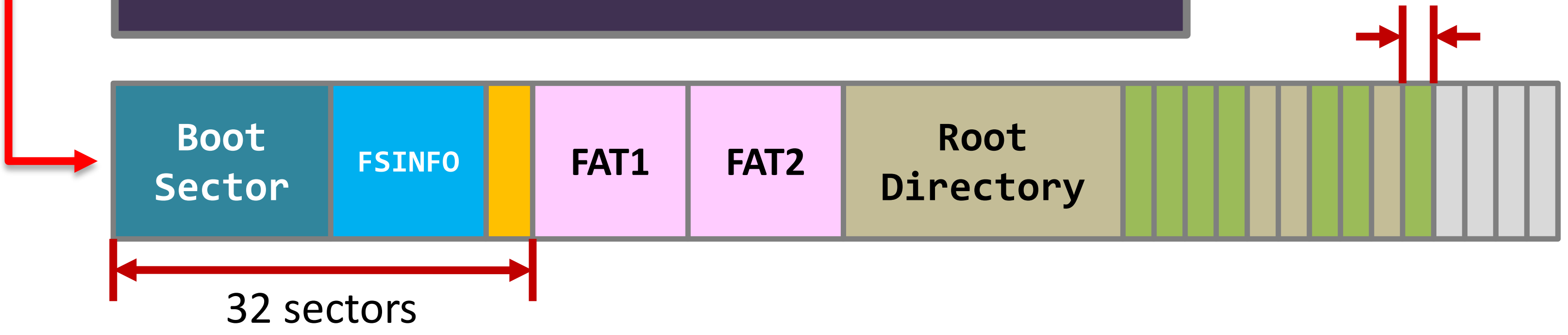


```
$ sudo mkfs.vfat -F32 /dev/ram0
mkfs.fat 3.0.28 (2015-05-16)
......
$ sudo dosfsck -v /dev/ram0
fsck.fat 3.0.28 (2015-05-16)

Checking we can access the last sector of the filesystem
Boot sector contents:
System ID "mkfs.fat"
Media byte 0xf8 (hard disk)
       512 bytes per logical sector
       512 bytes per cluster
        32 reserved sectors
First FAT starts at byte 16384 (sector 32)
         2 FATs, 32 bit entries
    516608 bytes per FAT (= 1009 sectors)
Root directory start at cluster 2 (arbitrary size)
Data area starts at byte 1049600 (sector 2050)
    129022 data clusters (66059264 bytes)
......
```

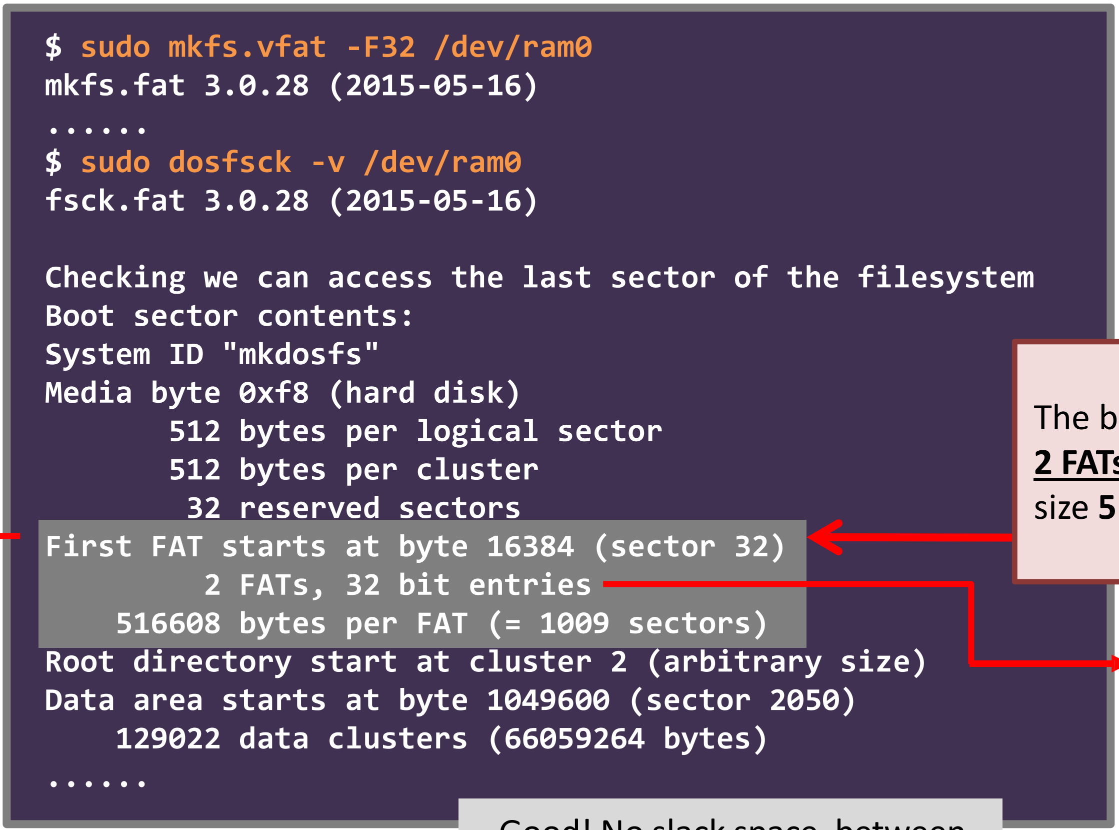
Details of the **Boot Sector**

The boot sector says:
A cluster is made of 1 sector.

One cluster size: 512 bytes in this case



Typical layout of a FAT32 partition

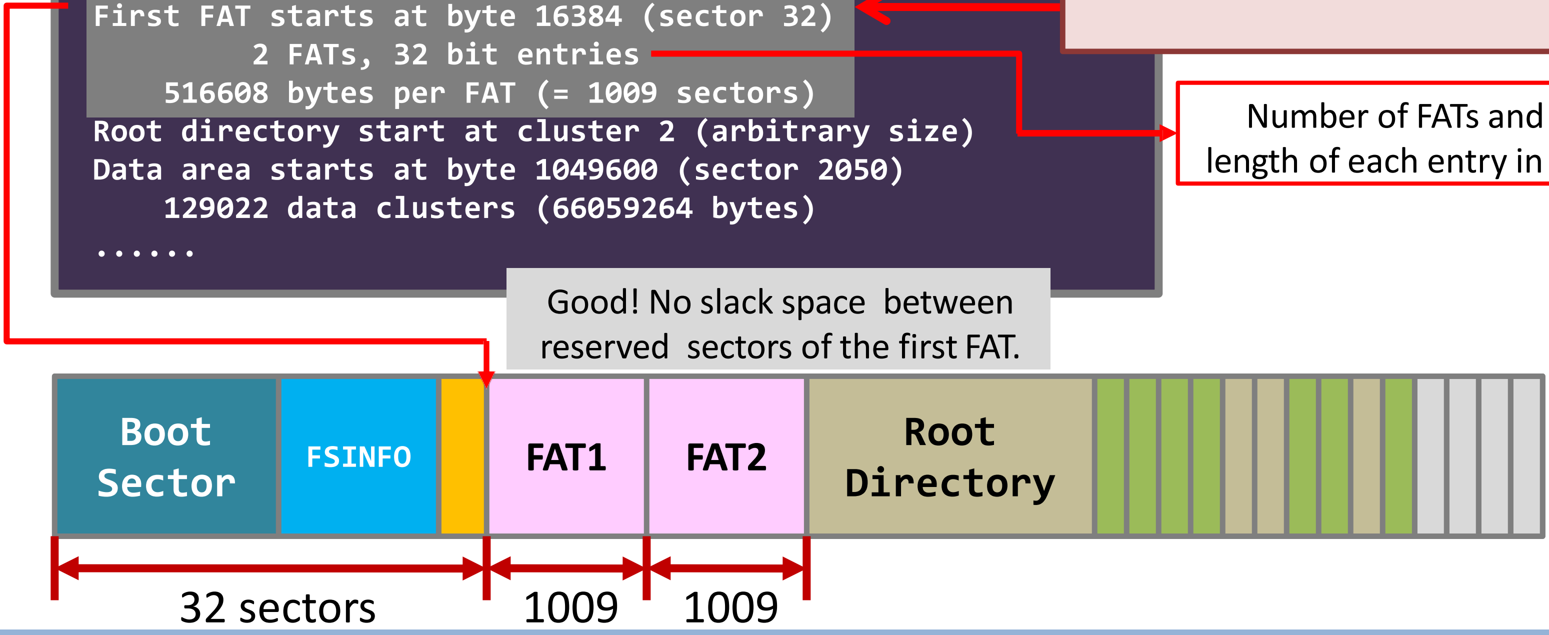


```
$ sudo mkfs.vfat -F32 /dev/ram0
mkfs.fat 3.0.28 (2015-05-16)
......
$ sudo dosfsck -v /dev/ram0
fsck.fat 3.0.28 (2015-05-16)

Checking we can access the last sector of the filesystem
Boot sector contents:
System ID "mkdosfs"
Media byte 0xf8 (hard disk)
       512 bytes per logical sector
       512 bytes per cluster
        32 reserved sectors
First FAT starts at byte 16384 (sector 32)
         2 FATs, 32 bit entries
    516608 bytes per FAT (= 1009 sectors)
Root directory start at cluster 2 (arbitrary size)
Data area starts at byte 1049600 (sector 2050)
    129022 data clusters (66059264 bytes)
......
```

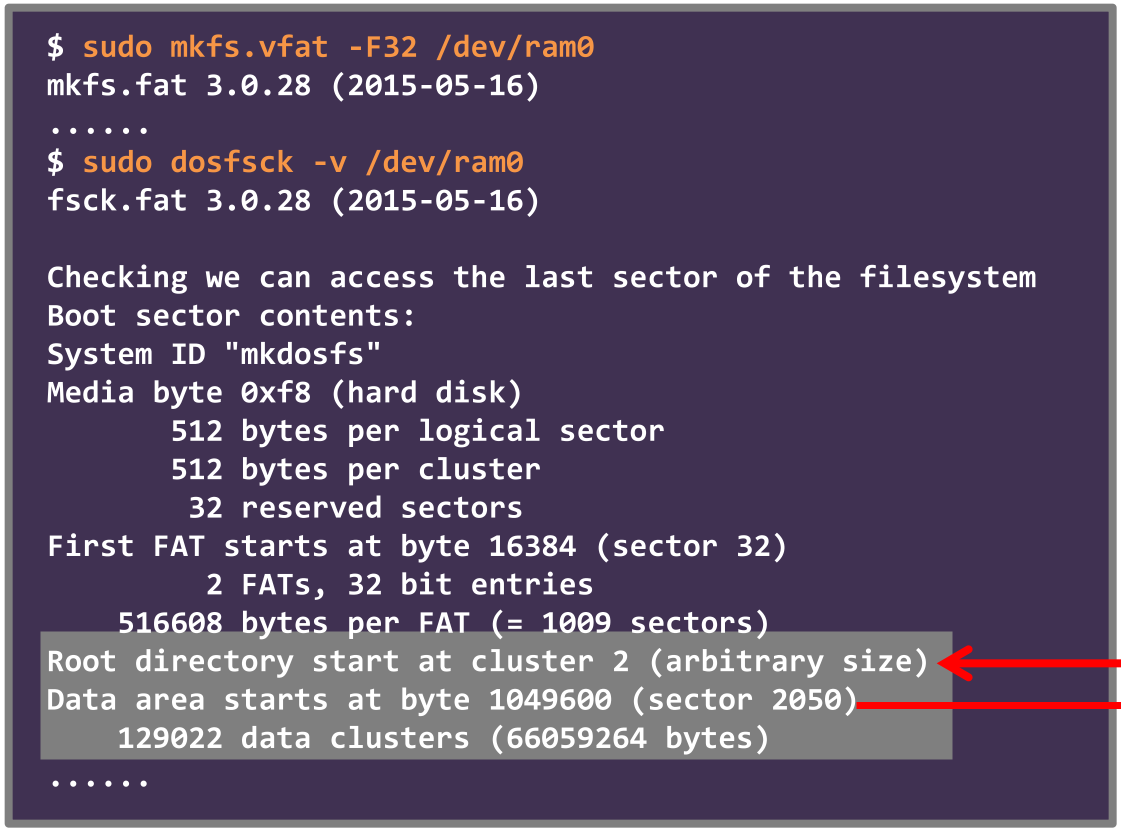
The boot sector says:
2 FATs and each of them is of size **516,608 bytes**.

Number of FATs and the length of each entry in a FAT.

Good! No slack space between reserved sectors of the first FAT.



Typical layout of a FAT32 partition

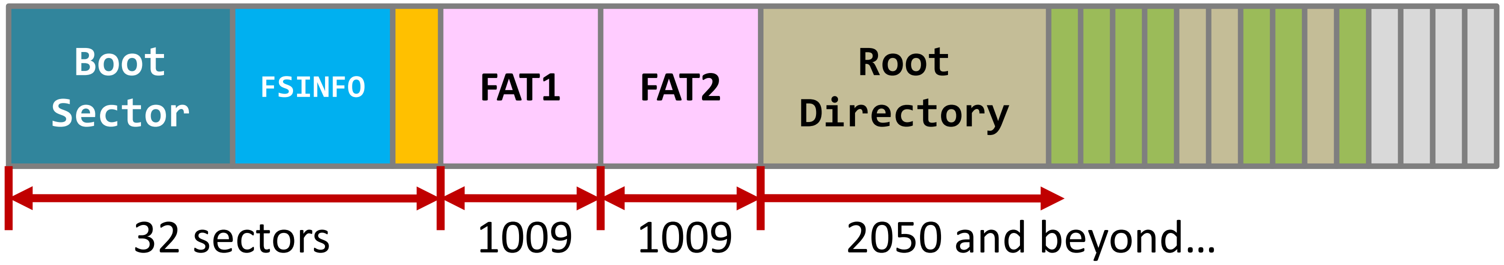


```
$ sudo mkfs.vfat -F32 /dev/ram0
mkfs.fat 3.0.28 (2015-05-16)
......
$ sudo dosfsck -v /dev/ram0
fsck.fat 3.0.28 (2015-05-16)

Checking we can access the last sector of the filesystem
Boot sector contents:
System ID "mkdosfs"
Media byte 0xf8 (hard disk)
       512 bytes per logical sector
       512 bytes per cluster
        32 reserved sectors
First FAT starts at byte 16384 (sector 32)
         2 FATs, 32 bit entries
    516608 bytes per FAT (= 1009 sectors)
Root directory start at cluster 2 (arbitrary size)
Data area starts at byte 1049600 (sector 2050)
    129022 data clusters (66059264 bytes)
......
```

The first data cluster is **Cluster #2** and it is usually, not always, the root directory.

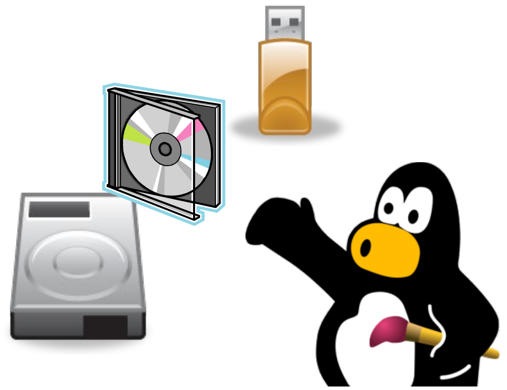
Cluster #0 & #1 are reserved.

`32 + 1009 x 2 = 2050`



Details of FAT32

- Introduction
- **Directory and File Attributes**
- File Operations
  - Read files
  - Write files
  - Delete files
  - Recover deleted files



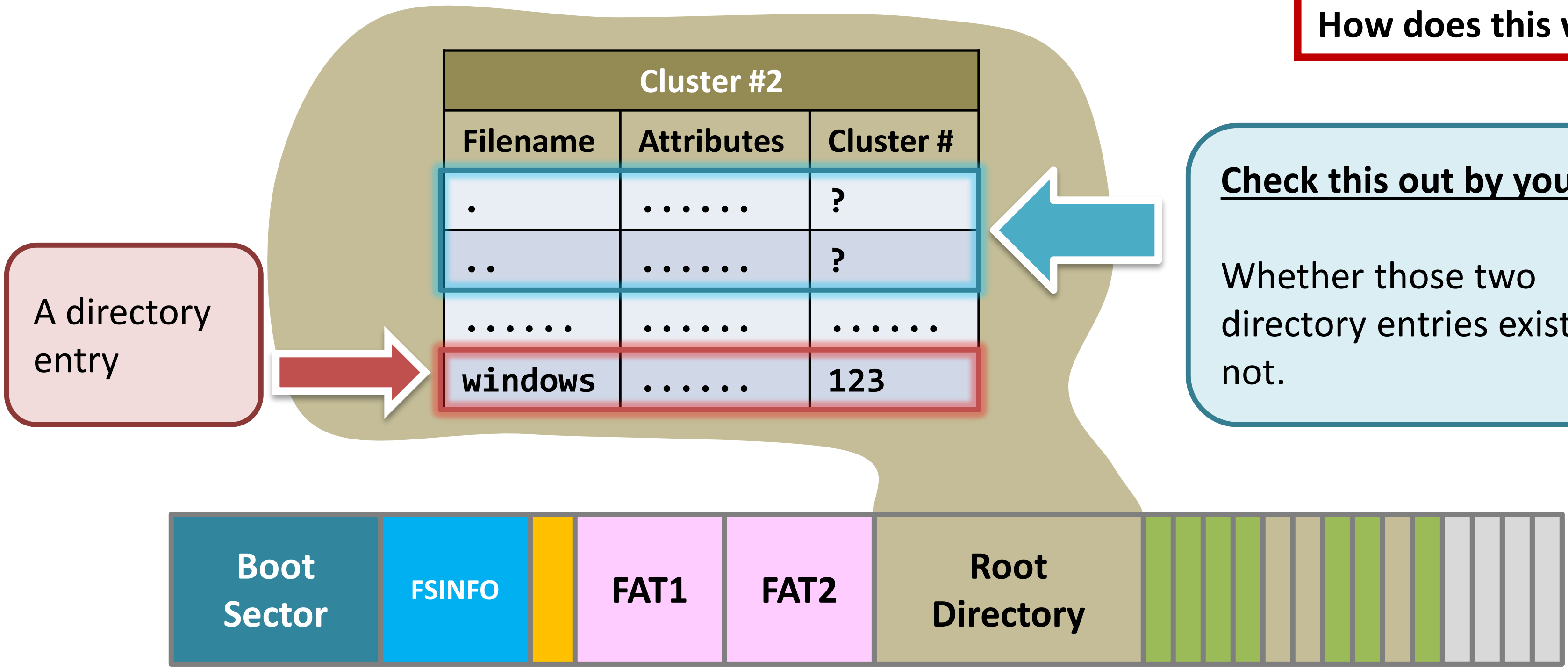
Directory Traversal

Step (1) Read the directory file of the root directory starting from **Cluster #2**.

"**C:\windows**" starts from Cluster #123.

```
c:\> dir c:\windows
……
06/13/2012  2,033,216   explorer.exe
08/04/2015    169,120   notepad.exe
……
c:\> _
```

How does this work?



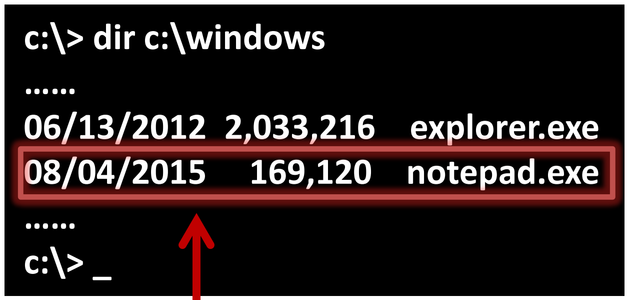
Cluster #2		
Filename	**Attributes**	**Cluster #**
.	......	?
..	......	?
......	......	......
windows	......	123

Check this out by yourself.

Whether those two directory entries exist or not.

Directory Traversal

Step (2) Read the directory file of the "**C:\windows**" starting from **Cluster #123**.

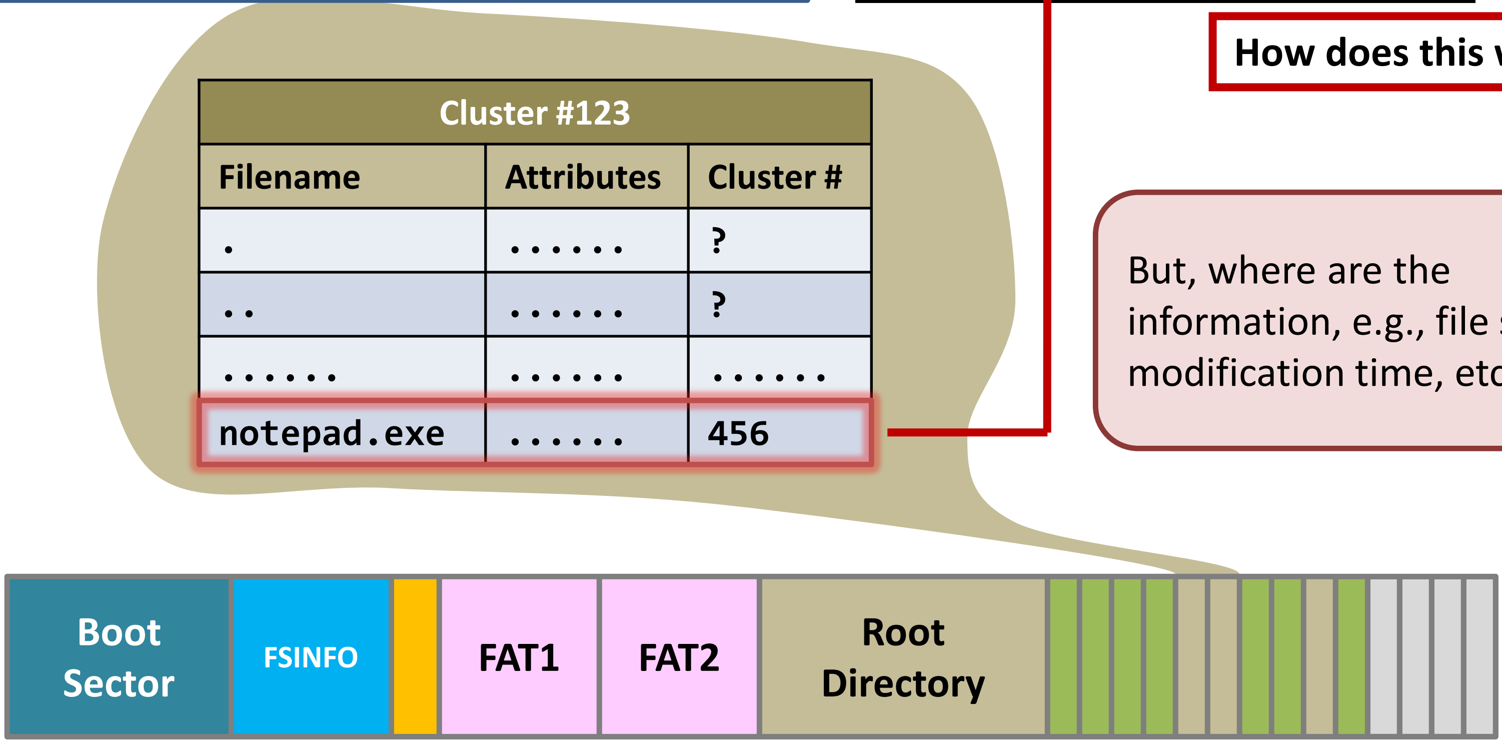


```
c:\> dir c:\windows
......
06/13/2012  2,033,216   explorer.exe
08/04/2015    169,120   notepad.exe
......
c:\> _
```

How does this work?

Cluster #123		
Filename	**Attributes**	**Cluster #**
.	......	?
..	......	?
......	......	......
notepad.exe	......	456

But, where are the information, e.g., file size, modification time, etc?



Directory entry

- Directory entry is just a structure.

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

what?

Filename	Attributes	Cluster #
explorer.exe	......	32

How?

0	e	x	p	l	o	r	e	r	7
8	e	x	e	…	…	…	…	…	15
16	…	…	…	…	00	00	…	…	23
24	…	…	20	00	00	C4	0F	00	31

Note. This is the 8+3 naming convention.

8 characters for name +
3 characters for file extension

Directory entry

- Directory entry is just a structure.

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

what?

Filename	Attributes	Cluster #
explorer.exe	......	32

How?

0	e	x	p	l	o	r	e	r	7
8	e	x	e	…	…	…	…	…	15
16	…	…	…	…	00	00	…	…	23
24	…	…	20	00	00	C4	0F	00	31

How to calculate the first cluster address?

Directory entry

- Directory entry is just a structure.

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

what?

Filename	Attributes	Cluster #
`explorer.exe`	`......`	32

How?

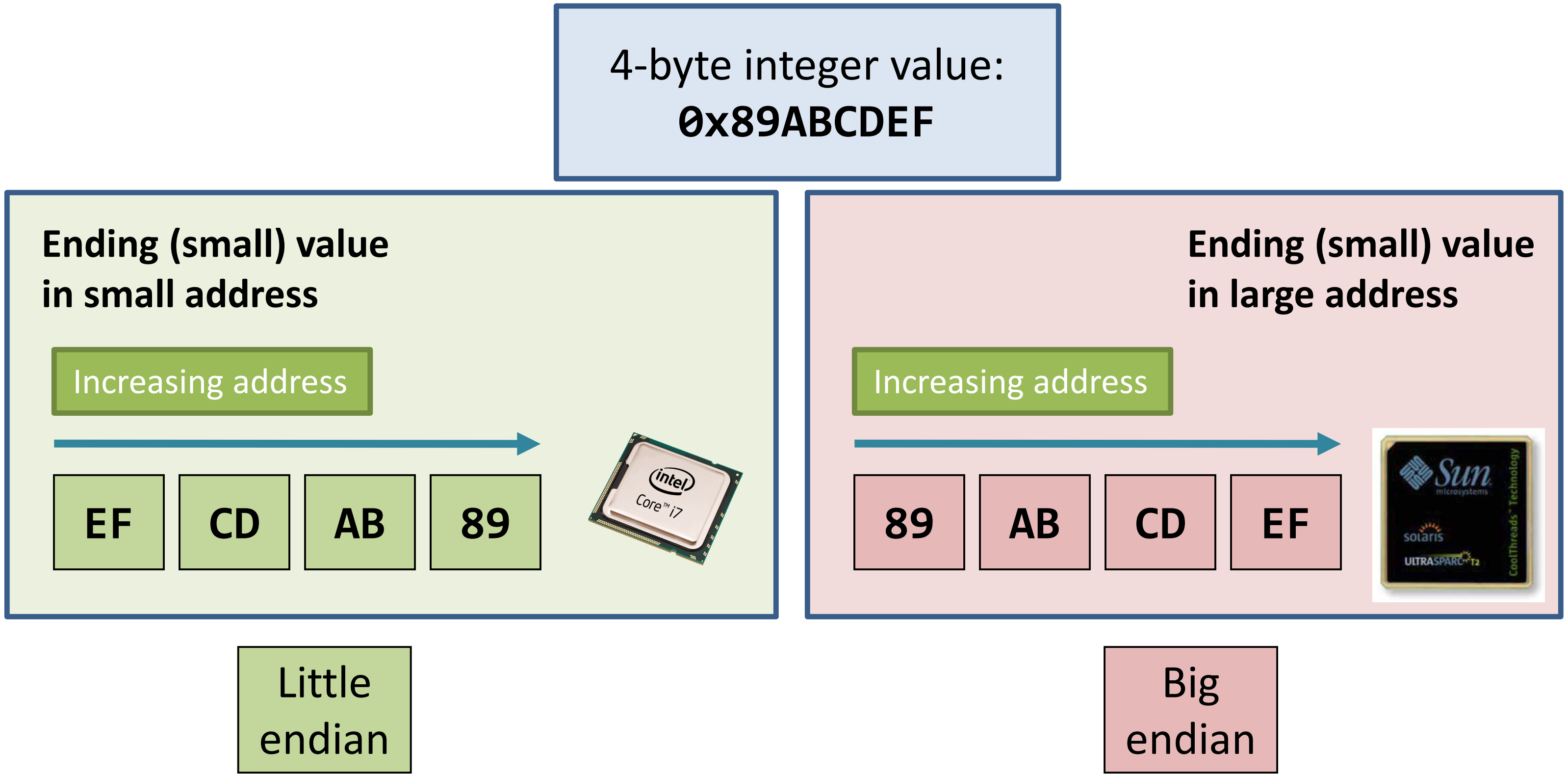
0	e	x	p	l	o	r	e	r	7
8	e	x	e	…	…	…	…	…	15
16	…	…	…	…	00	00	…	…	23
24	…	…	20	00	00	C4	0F	00	31

Higher 2 bytes	Lower 2 bytes		Cluster address
00 00	20 00	=	8192

It is not 32, why?

Big Endian vs Little Endian

- Endian-ness is about byte ordering.
  - It means the way that a machine (we mean the entire computer architecture) orders the bytes.



Big Endian vs Little Endian

- Directory entry is just a structure.

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

what?

Filename	Attributes	Cluster #
`explorer.exe`	`......`	32

How?

0	e	x	p	l	o	r	e	r	7
8	e	x	e	…	…	…	…	…	15
16	…	…	…	…	00	00	…	…	23
24	…	…	20	00	00	C4	0F	00	31

Big endian	00	00	20	00	=	8192
Little endian	00	00	00	20	=	32

The FAT is defined to use little-endian byte ordering, as its original implementation was on the Intel x86 platform

The file size…

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

what?

Filename	Attributes	Cluster #
explorer.exe	......	32

How?

0	e	x	p	l	o	r	e	r	7
8	e	x	e	…	…	…	…	…	15
16	…	…	…	…	00	00	…	…	23
24	…	…	20	00	00	C4	0F	00	31

So, what is the largest size of a file?

4G – 1 bytes

Directory entry

- Any problem with this design?

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

Note. This is the 8+3 naming convention.

8 characters for name +
3 characters for file extension

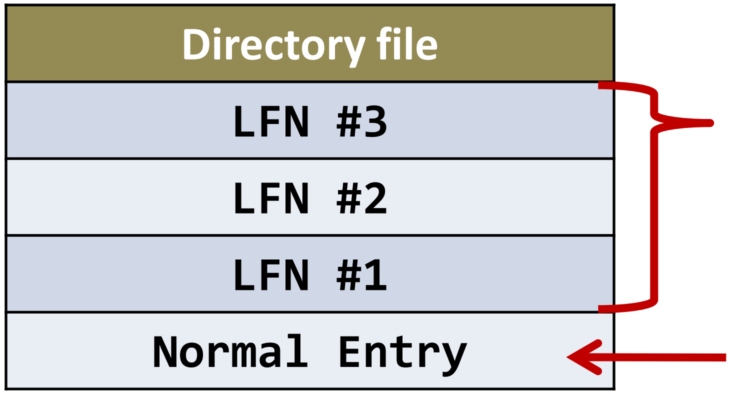
Example:

How to store the file:
"I_love_the_operating_system_course.txt"

How to store long filename?

FAT series – LFN directory entry

- LFN: Long File Name.
  - In FAT32, the 8+3 naming convention is removed by…
  - Adding more entries to represent the filename



Each LFN entry represents 13 characters in Unicode, i.e., 2 bytes per character.
Yet, the sequence is upside-down!

The normal directory entry is still there.

FAT series – LFN directory entry

Normal entry

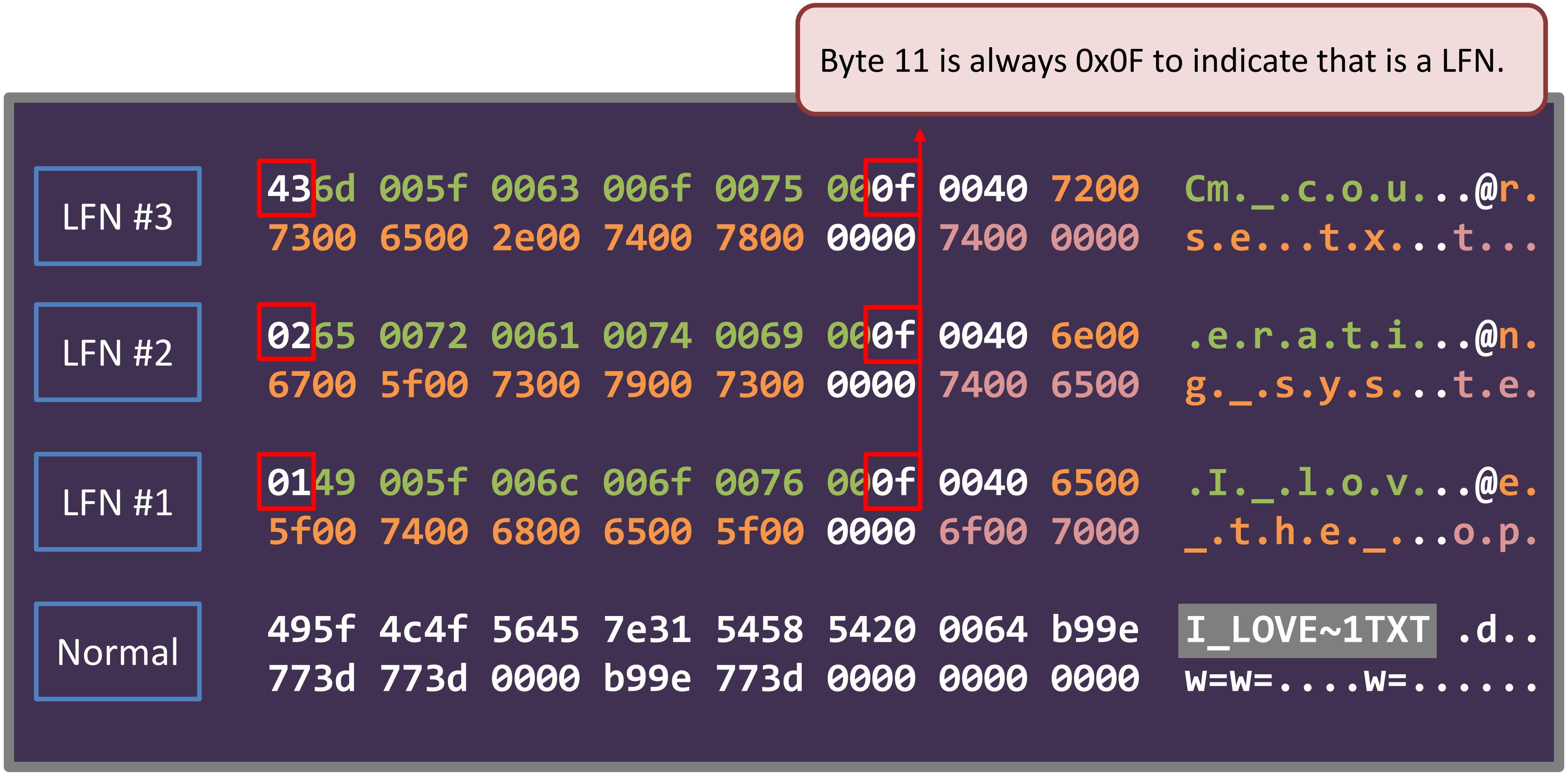
Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)
1-10	7+3 characters of filename + extension.
11-11	File attributes (e.g., read only, hidden)
12-12	Reserved.
13-19	Creation and access time information.
20-21	High 2 bytes of the first cluster address (0 for FAT16 and FAT12).
22-25	Written time information.
26-27	Low 2 bytes of first cluster address.
28-31	File size.

LFN entry

Bytes	Description
0-0	Sequence Number
1-10	File name characters (5 characters in Unicode)
11-11	File attributes - always 0x0F
12-12	Reserved.
13-13	Checksum
14-25	File name characters (6 characters in Unicode)
26-27	Reserved
28-31	File name characters (2 characters in Unicode)

FAT series – LFN directory entry

- Filename: “`I_love_the_operating_system_course.txt`”.



Byte 11 is always 0x0F to indicate that is a LFN.

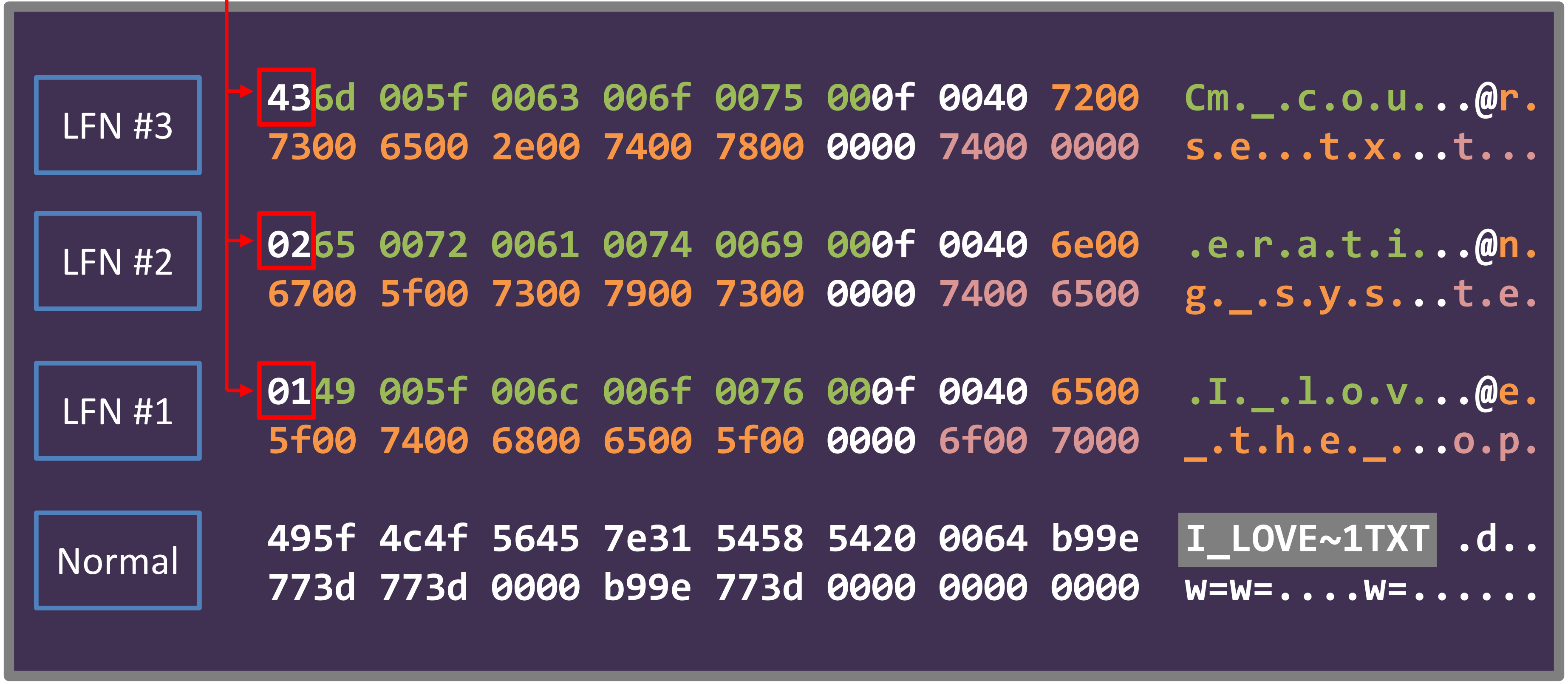
```
4336d 005f 0063 006f 0075 000f 0040 7200  Cm._.c.o.u...@r.
7300 6500 2e00 7400 7800 0000 7400 0000  s.e...t.x...t...

0265 0072 0061 0074 0069 000f 0040 6e00  .e.r.a.t.i...@n.
6700 5f00 7300 7900 7300 0000 7400 6500  g._.s.y.s...t.e.

0149 005f 006c 006f 0076 000f 0040 6500  .I._.l.o.v...@e.
5f00 7400 6800 6500 5f00 0000 6f00 7000  _.t.h.e._...o.p.

495f 4c4f 5645 7e31 5458 5420 0064 b99e  I_LOVE~1TXT .d..
773d 773d 0000 b99e 773d 0000 0000 0000  w=w=....w=......
```

FAT series – LFN directory entry

This is the sequence number, and they are arranged in descending order.

The terminating directory entry has the sequence number **OR-ed with 0x40**.

Directory file
LFN #3: "m_cou" "rse.tx" "t"
LFN #2: "erati" "ng_sys" "te"
LFN #1: "I_lov" "e_the_" "op"
Normal Entry



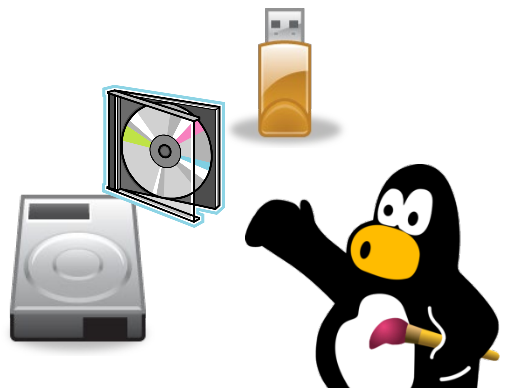
```
LFN #3   436d 005f 0063 006f 0075 000f 0040 7200  Cm._.c.o.u...@r.
         7300 6500 2e00 7400 7800 0000 7400 0000  s.e...t.x...t...

LFN #2   0265 0072 0061 0074 0069 000f 0040 6e00  .e.r.a.t.i...@n.
         6700 5f00 7300 7900 7300 0000 7400 6500  g._.s.y.s...t.e.

LFN #1   0149 005f 006c 006f 0076 000f 0040 6500  .I._.l.o.v...@e.
         5f00 7400 6800 6500 5f00 0000 6f00 7000  _.t.h.e._...o.p.

Normal   495f 4c4f 5645 7e31 5458 5420 0064 b99e  I_LOVE~1TXT .d..
         773d 773d 0000 b99e 773d 0000 0000 0000  w=w=....w=......
```

FAT series – directory entry: a short summary

- A directory is an extremely important part of a FAT-like file system.
  - It stores the **start of the content**, i.e., the start cluster number.
  - It stores the **end of the content**, i.e., the **file size**; without the file size, how can you know when you should stop reading a cluster?
  - It stores **all file attributes**.

Details of FAT32

- Introduction
- Directory and File Attributes
- File Operations
  - **Read files**
  - Write files
  - Delete files
  - Recover deleted files



How to read a file?

Task: read "C:\windows\explorer.exe" sequentially.

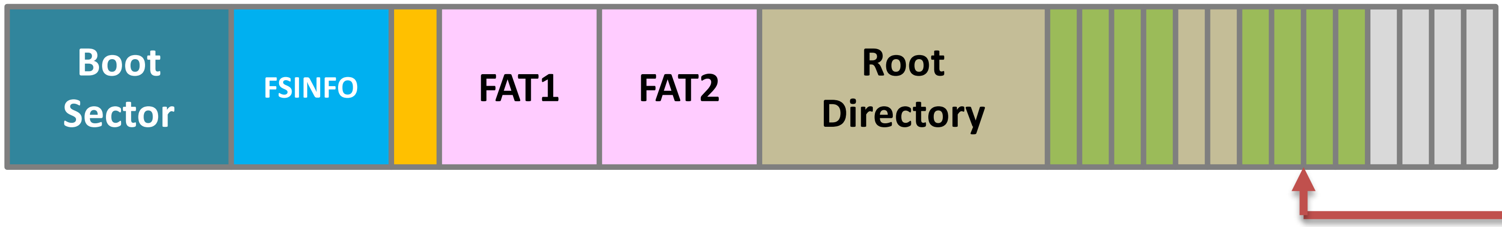
Suppose we already read out the directory entry…

You know the process of directory traversal, right?

Filename	Attributes	Cluster #
`explorer.exe`	....	32

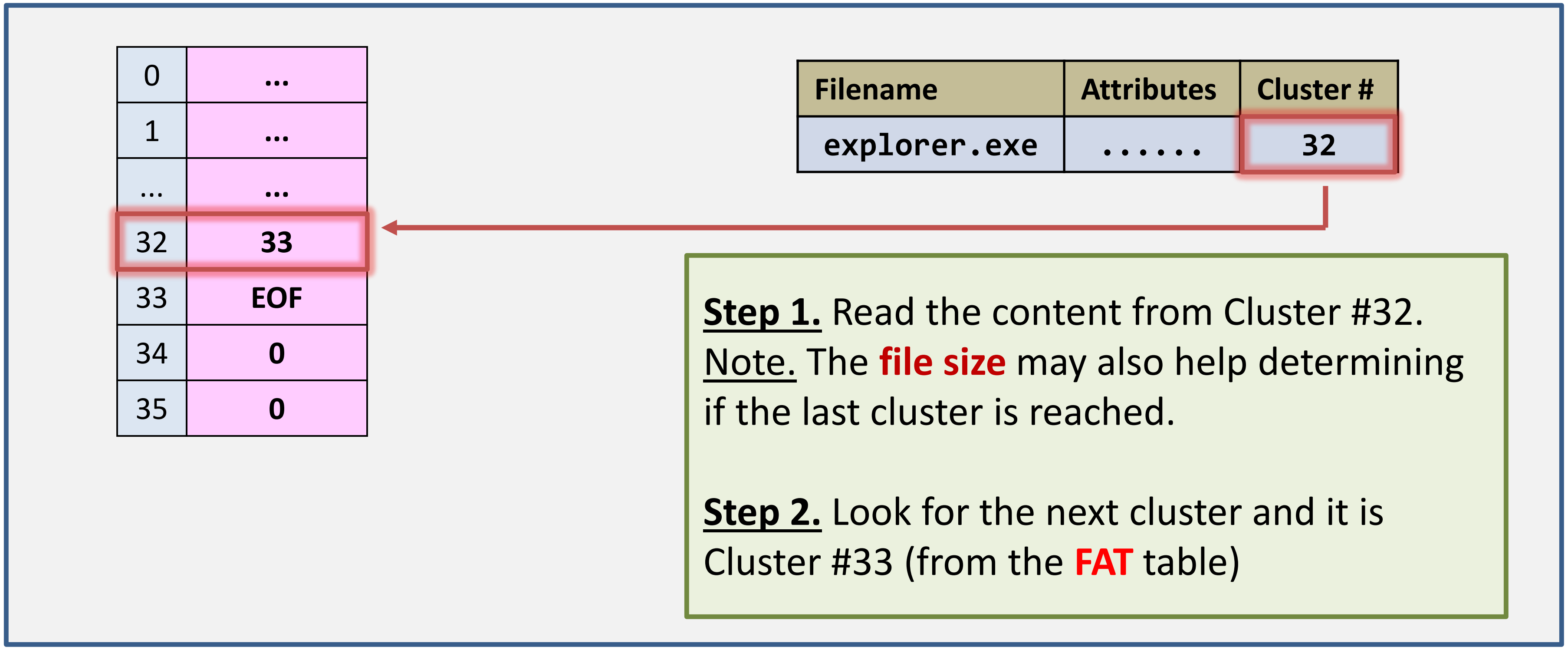
Step 1. Read the content from Cluster #32.

Note. The **file size** may also help determining if the last cluster is reached (remember where it is stored?)



How to read a file?

Task: read "C:\windows\explorer.exe" sequentially.

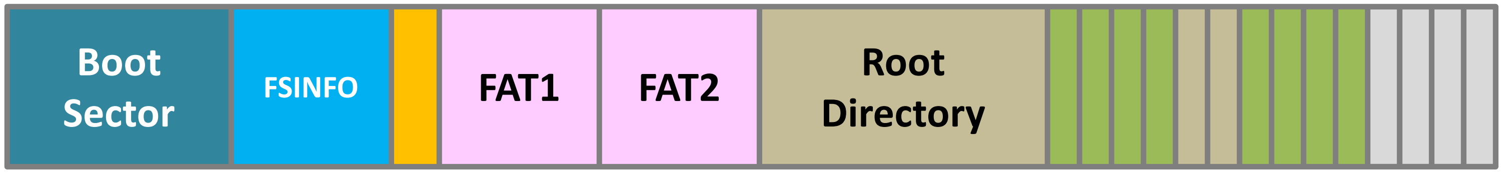


0	…
1	…
…	…
32	**33**
33	**EOF**
34	**0**
35	**0**

Filename	Attributes	Cluster #
`explorer.exe`	`......`	32

<u>Step 1.</u> Read the content from Cluster #32.
<u>Note.</u> The **file size** may also help determining if the last cluster is reached.

<u>Step 2.</u> Look for the next cluster and it is Cluster #33 (from the **FAT** table)



How to read a file?

Task: read "C:\windows\explorer.exe" sequentially.

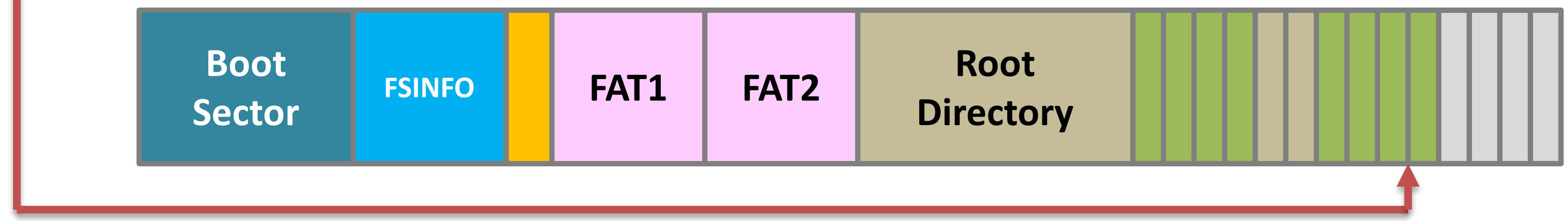
0	…
1	…
…	…
32	**33**
33	**EOF**
34	**0**
35	**0**

Filename	Attributes	Cluster #
`explorer.exe`	....	32

Step 3. Since the FAT has marked "EOF", we have reached the last cluster.

Note. The file size help determine **how many bytes to read** from the last cluster.

FAT entry structure??
Remember: 28bits are used to represent cluster number for FAT32



How to read a file?

Task: read "C:\windows\explorer.exe" sequentially.

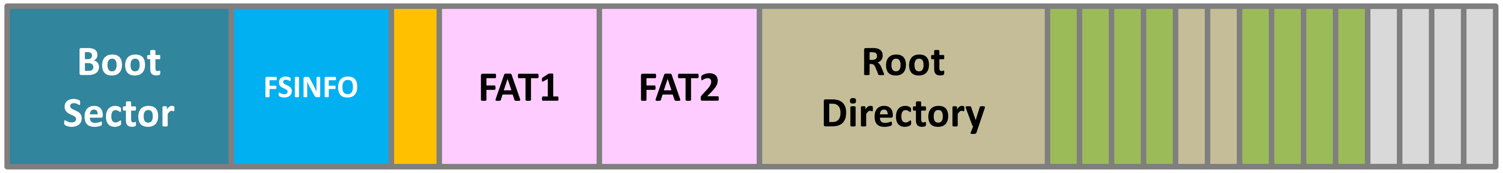
0	…
1	…
…	…
32	**33**
33	**EOF**
34	**0**
35	**0**

Damaged	**= 0x0ffffff7**
EOF	**>= 0x0ffffff8**
Unallocated	**= 0x0**

Filename	Attributes	Cluster #
`explorer.exe`	`......`	`32`

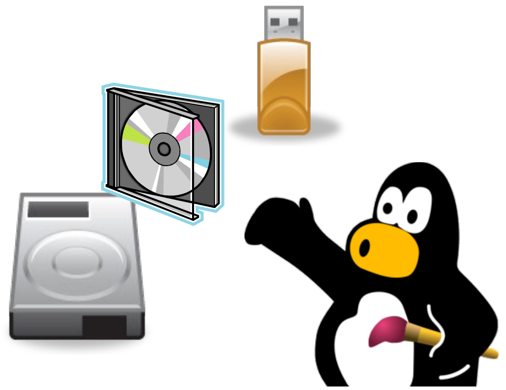
Step 3. Since the FAT has marked "EOF", we have reached the last cluster.

Note. The file size help determine **how many bytes to read** from the last cluster.



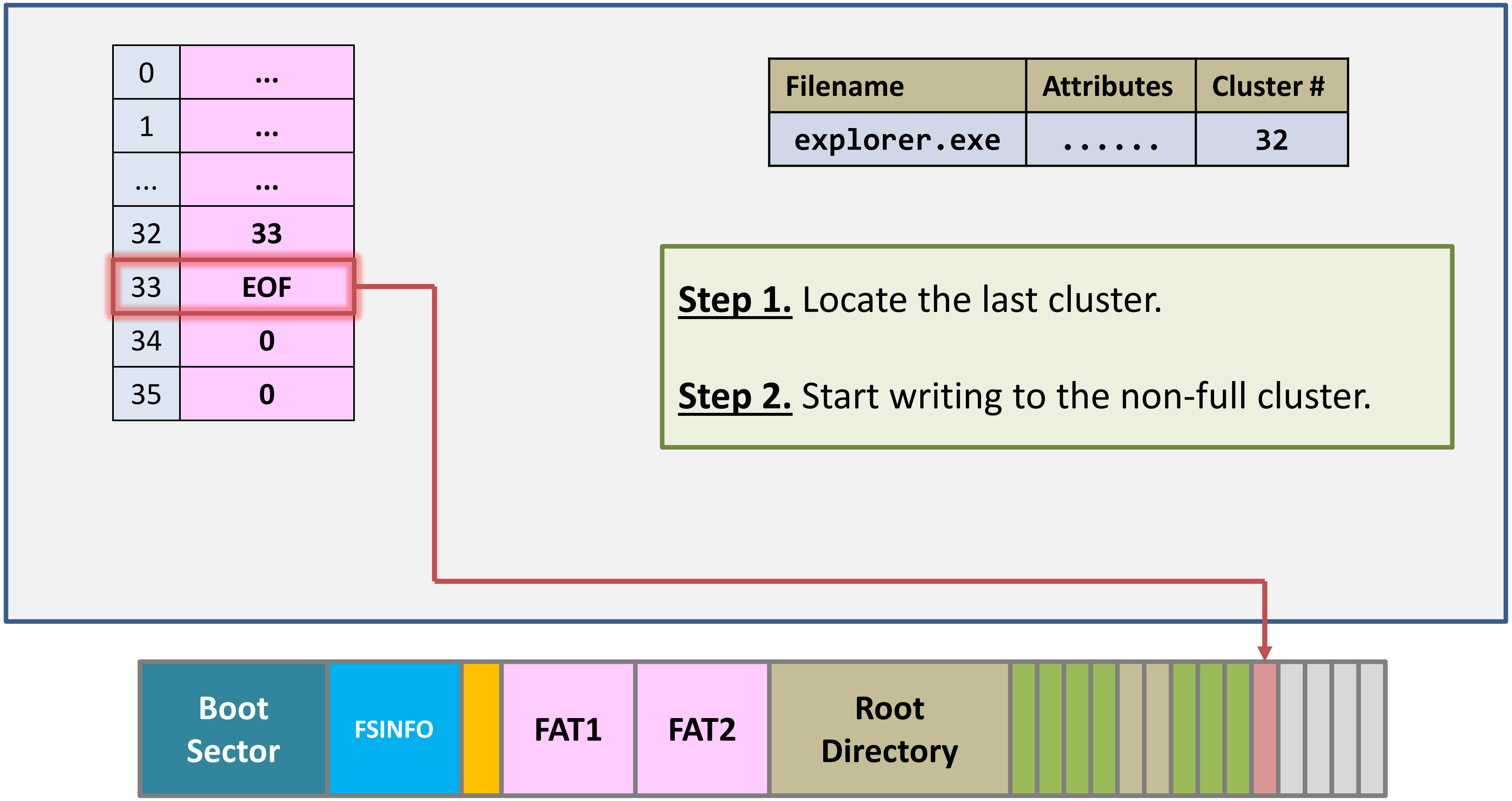
Details of FAT32

- Introduction
- Directory and File Attributes
- File Operations
  - Read files
  - **Write files**
  - Delete files
  - Recover deleted files



How to write a file?

Task: append data to "C:\windows\explorer.exe".



0	…
1	…
…	…
32	**33**
33	**EOF**
34	**0**
35	**0**

Filename	Attributes	Cluster #
`explorer.exe`	`......`	`32`

Step 1. Locate the last cluster.

Step 2. Start writing to the non-full cluster.

How to write a file?

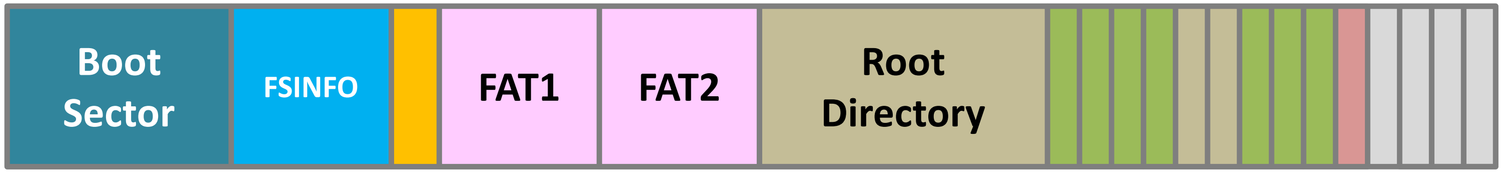
Task: append data to "C:\windows\explorer.exe".

0	**…**
1	**…**
…	**…**
32	**33**
33	**EOF**
34	**0**
35	**0**

Filename	**Attributes**	**Cluster #**
`explorer.exe`	....	32

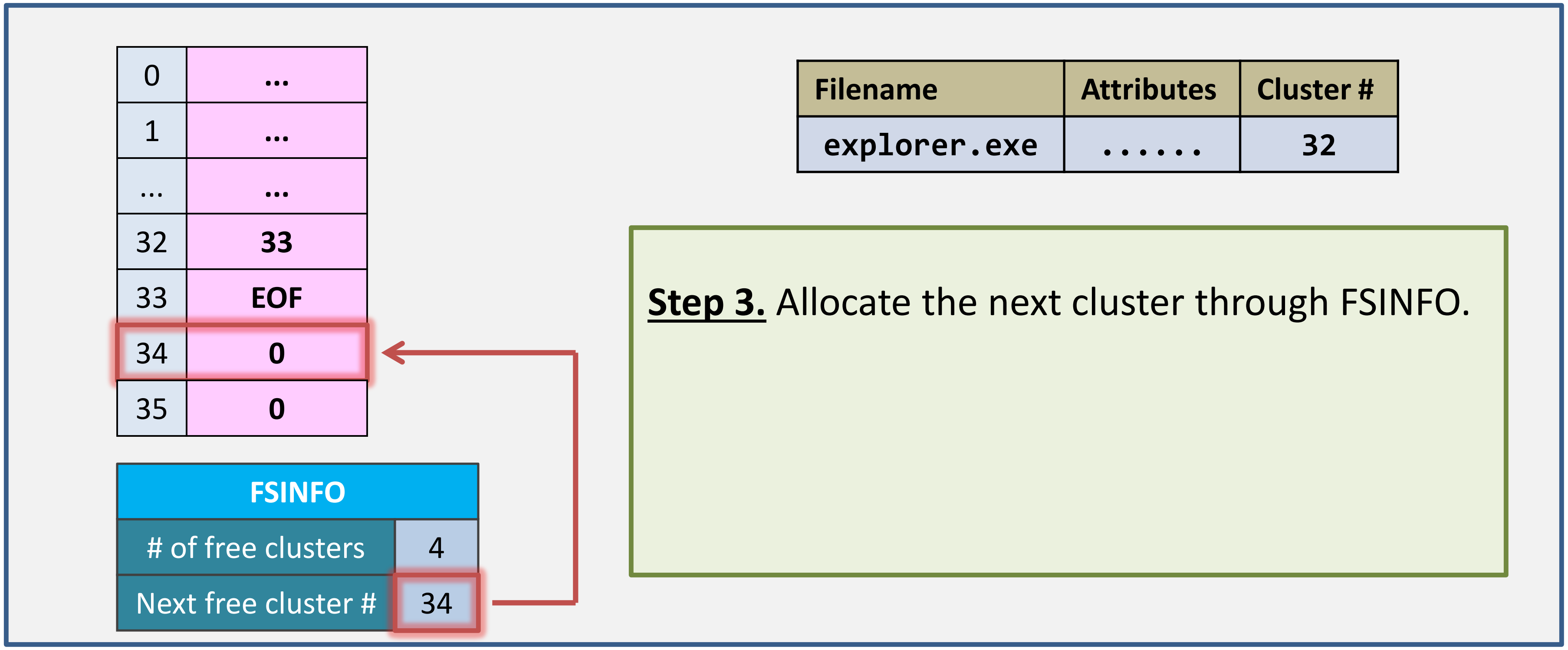
Step 3. Allocate the next cluster through FSINFO.

What is stored in FSINFO?
How to allocate?



How to write a file?

Task: append data to "C:\windows\explorer.exe".

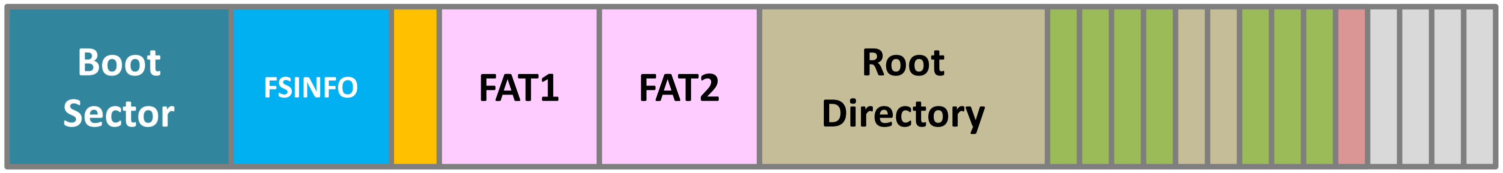


0	…
1	…
…	…
32	**33**
33	**EOF**
34	**0**
35	**0**

Filename	Attributes	Cluster #
explorer.exe	…….	32

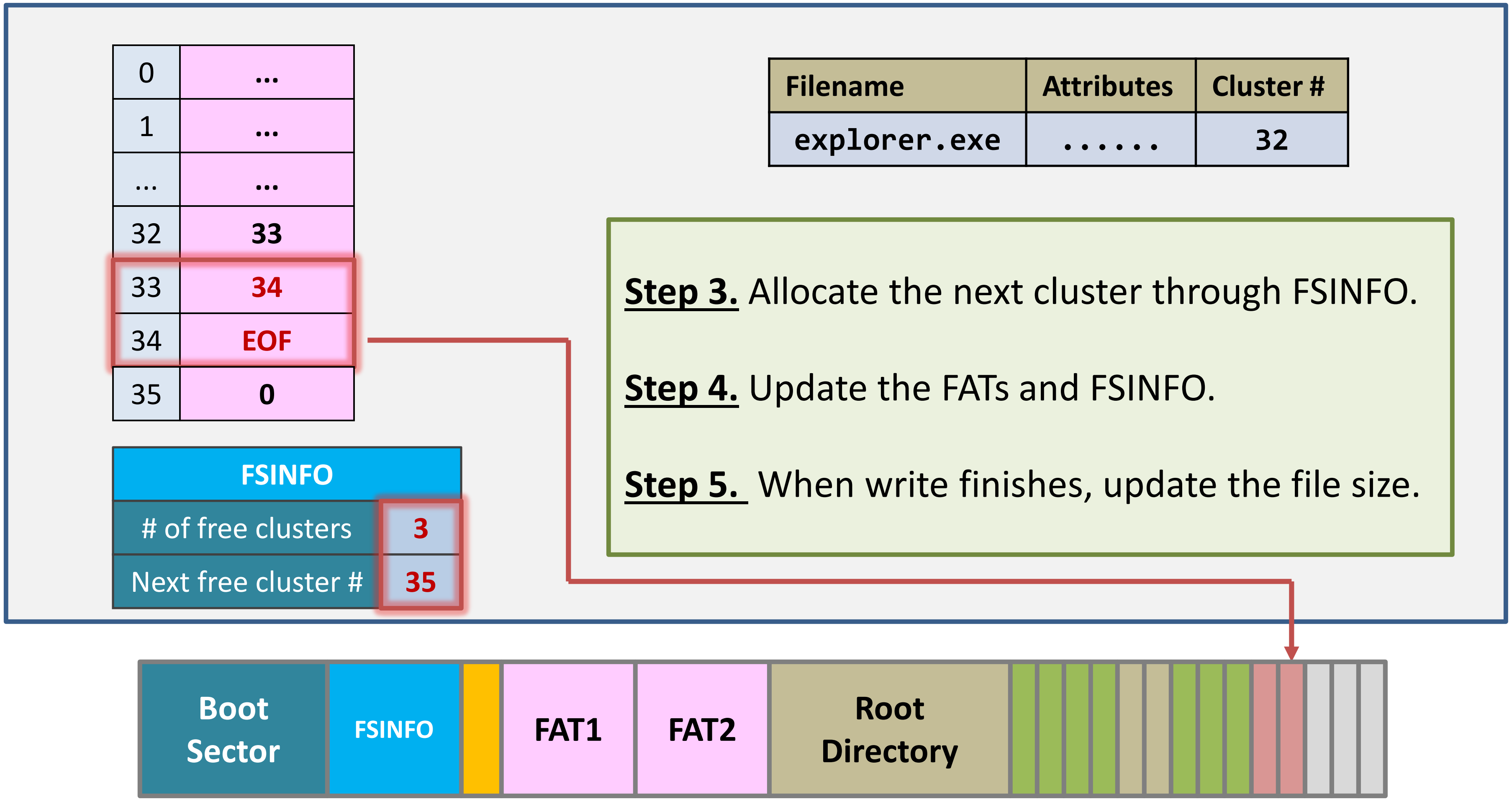
FSINFO	
# of free clusters	4
Next free cluster #	34

Step 3. Allocate the next cluster through FSINFO.



How to write a file?

Task: append data to "C:\windows\explorer.exe".



0	…
1	…
…	…
32	**33**
33	**34**
34	**EOF**
35	**0**

FSINFO	
# of free clusters	**3**
Next free cluster #	**35**

Filename	Attributes	Cluster #
explorer.exe	......	32

Step 3. Allocate the next cluster through FSINFO.

Step 4. Update the FATs and FSINFO.

Step 5. When write finishes, update the file size.

How to write a file?

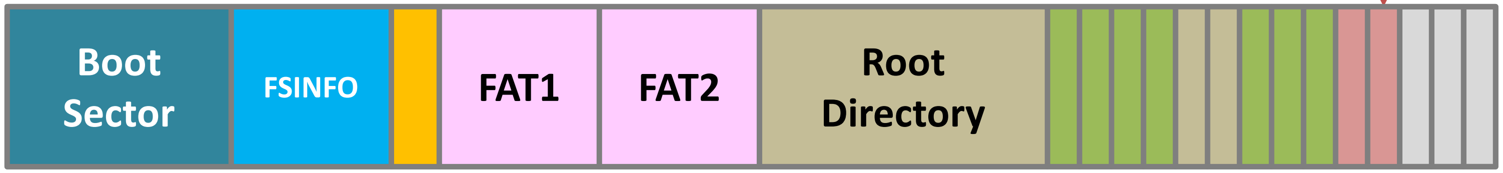
Task: append data to "C:\windows\explorer.exe".

0	…
1	…
…	…
32	**33**
33	**34**
34	**EOF**
35	**0**

FSINFO	
# of free clusters	**3**
Next free cluster #	**35**

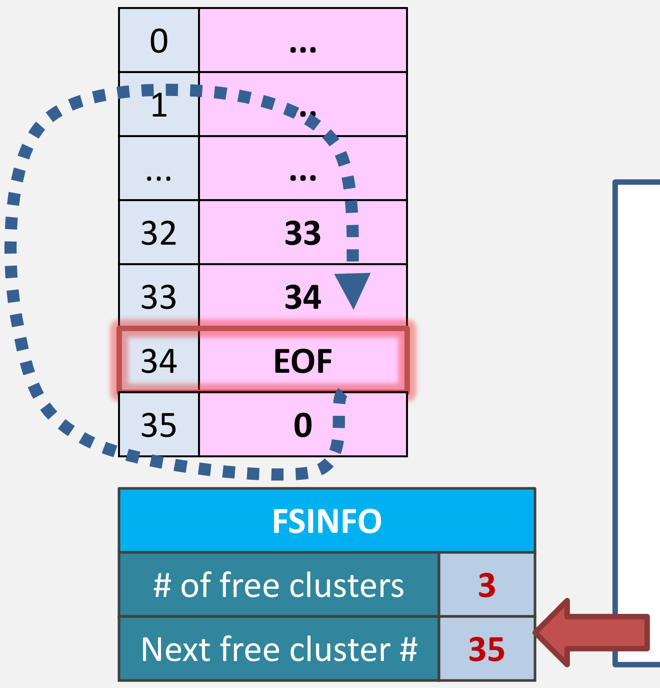
Filename	**Attributes**	**Cluster #**
`explorer.exe`	…….	32

Q: How to obtain the next free cluster?



How to write a file?

Task: append data to "C:\windows\explorer.exe".



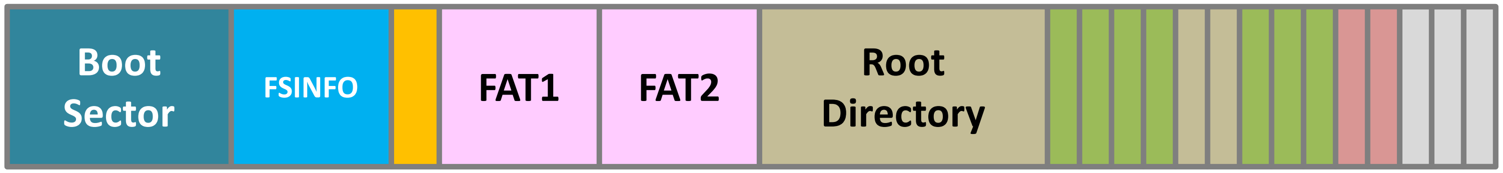
0	…
1	…
…	…
32	**33**
33	**34**
34	**EOF**
35	**0**

Filename	Attributes	Cluster #
explorer.exe	......	32

The search for the next free cluster is a **circular, next-available** search.

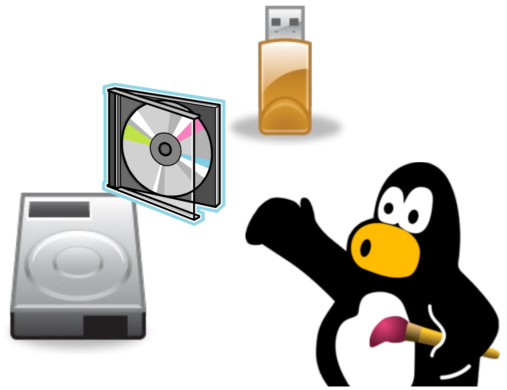
Why implementing next-available?
Principle of locality

Why circular?
To find out every free block



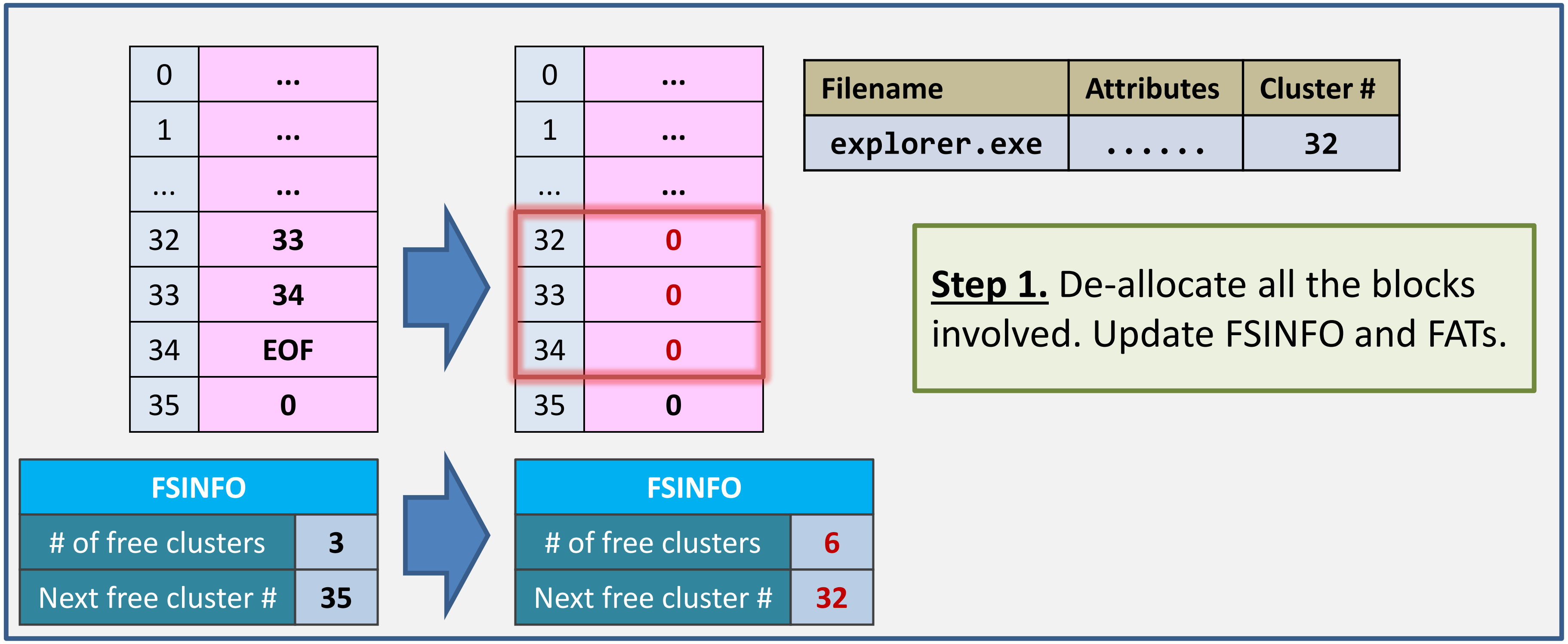
Details of FAT32

- Introduction
- Directory and File Attributes
- File Operations
  - Read files
  - Write files
  - **Delete files**
  - Recover deleted files



How to delete a file?

Task: delete "C:\windows\explorer.exe".



0	…
1	…
…	…
32	**33**
33	**34**
34	**EOF**
35	**0**

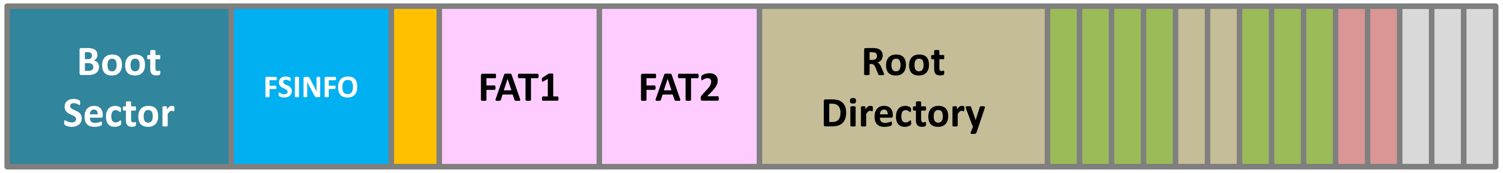
0	…
1	…
…	…
32	**0**
33	**0**
34	**0**
35	**0**

Filename	Attributes	Cluster #
`explorer.exe`	`......`	**32**

Step 1. De-allocate all the blocks involved. Update FSINFO and FATs.

FSINFO	
# of free clusters	**3**
Next free cluster #	**35**

FSINFO	
# of free clusters	**6**
Next free cluster #	**32**

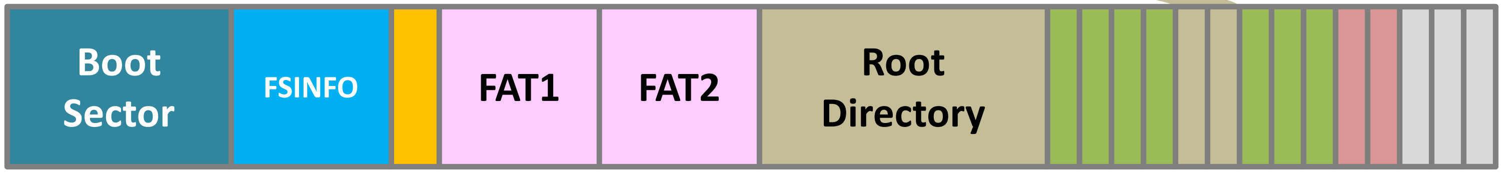


How to delete a file?

Task: delete “C:\windows\explorer.exe”.

How about the directory entry

Cluster #123		
Filename	Attributes	Cluster #
.	......	?
..	......	?
explorer.exe	......	32
notepad.exe	......	456



How to delete a file?

Task: delete "C:\windows\explorer.exe".

Bytes	Description
0-0	1st character of the filename (0x00 or 0xe5 means unallocated)

The first character becomes "0xE5".

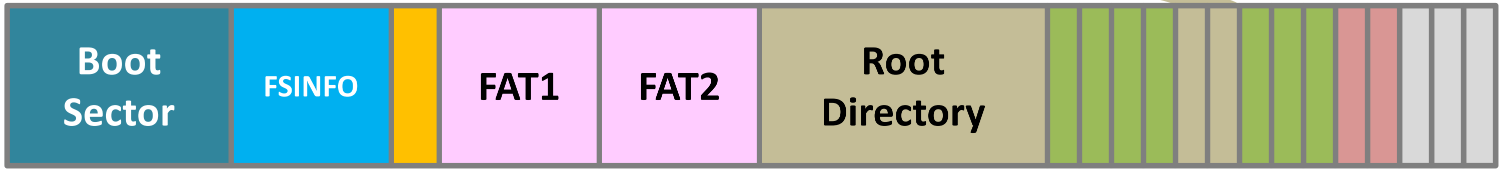
Cluster #123		
Filename	**Attributes**	**Cluster #**
.	......	?
..	......	?
_xplorer.exe	......	32
notepad.exe	......	456

How about the directory entry

Step 2. Change the first byte of the directory entry to 0xE5.

LFN entries also receive the same treatment.

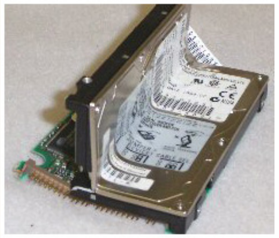
That's the end of deletion!



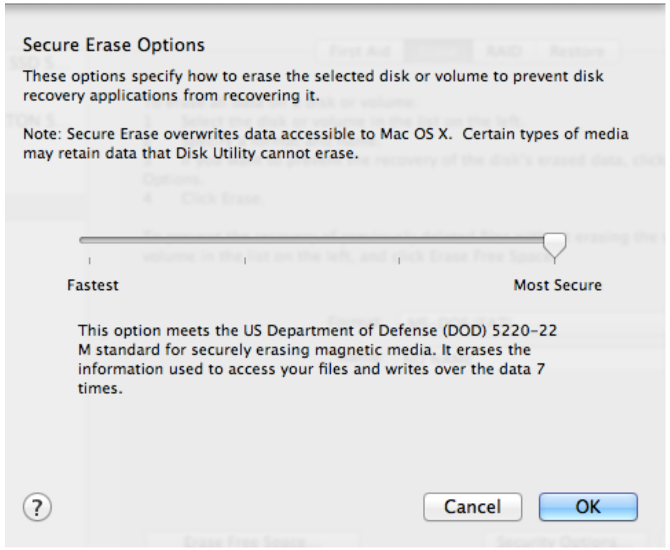
Really delete a file?

- Can you see that: **the file is not really removed from the FS layout?**
  - Perform a search in all the free space. Then, you will find all deleted file contents.
- "*Deleted data*" persists until the de-allocated clusters **are reused**.
  - This is an issue between performance (during deletion) and security.
- Any way(s) to delete a file **securely**?

How to delete a file “securely”?



Mac OS X Secure Disk Erase



Secure Erase Options

These options specify how to erase the selected disk or volume to prevent disk recovery applications from recovering it.

Note: Secure Erase overwrites data accessible to Mac OS X. Certain types of media may retain data that Disk Utility cannot erase.

This option meets the US Department of Defense (DOD) 5220-22 M standard for securely erasing magnetic media. It erases the information used to access your files and writes over the data 7 times.

Brute Force?

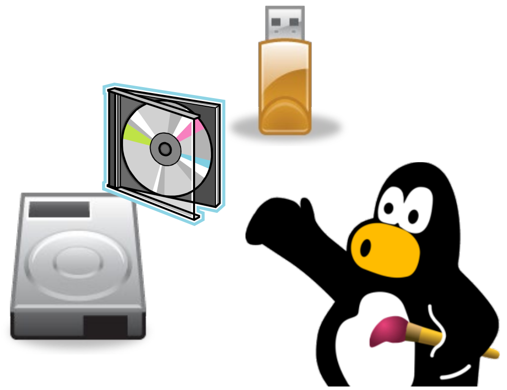
http://www.ohgizmo.com/2009/06/01/manual-hard-drive-destroyer-looks-like-fun/

What will the research community tell you?

http://cdn.computerscience1.net/2006/fall/lectures/8/articles8.pdf

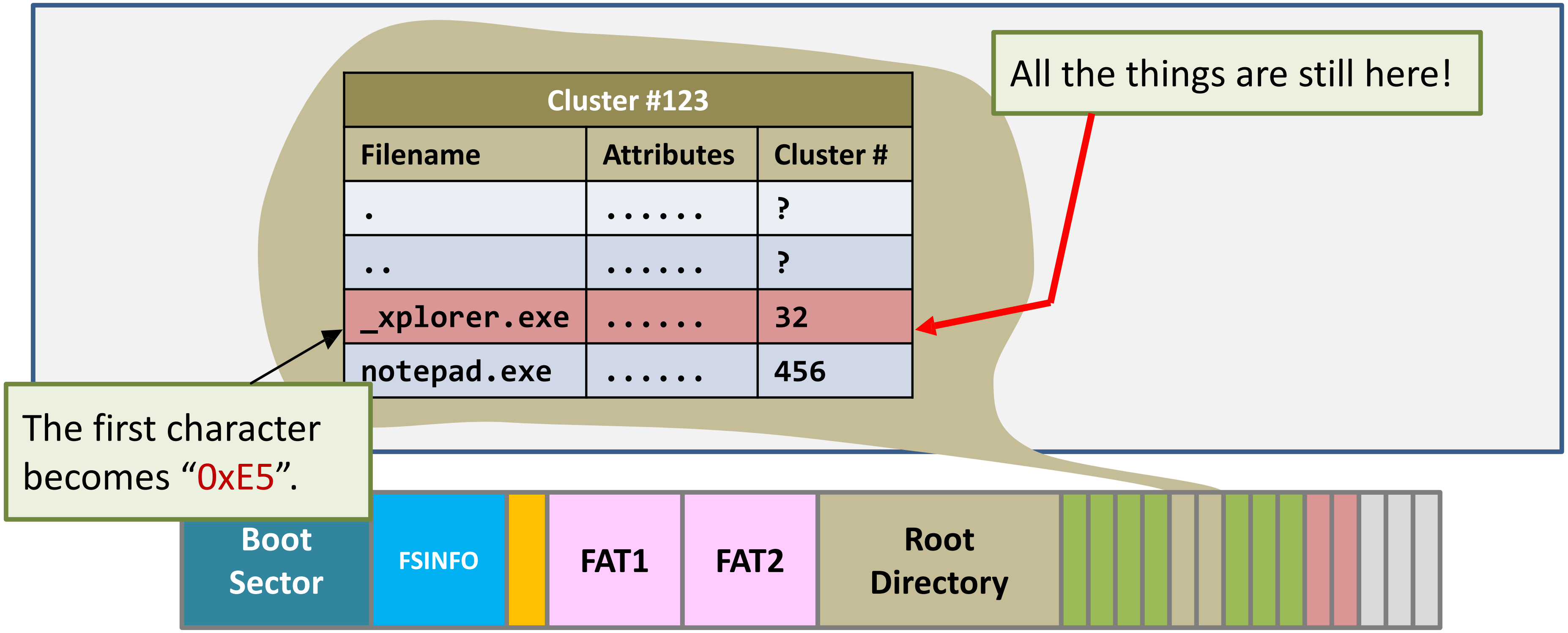
Details of FAT32

- Introduction
- Directory and File Attributes
- File Operations
  - Read files
  - Write files
  - Delete files
  - **Recover deleted files**



How to "*rescue*" a deleted file?

- If you're really care about the deleted file, then…
  - **PULL THE POWER PLUG AT ONCE!**
  - Pulling the power plug stops the target clusters from being over-written.



Cluster #123

Filename	Attributes	Cluster #
.	……	?
..	……	?
_xplorer.exe	……	32
notepad.exe	……	456

All the things are still here!
The first character becomes "0xE5".
Boot Sector
FSINFO
FAT1
FAT2
Root Directory

How to “*rescue*” a deleted file?

- If you’re really care about the deleted file, then…
  - **PULL THE POWER PLUG AT ONCE!**
  - Pulling the power plug stops the target clusters from being over-written.

<u>Principle of “rescue” deleted file</u>

Data persists unless the sectors are reallocated and overwritten.

File size <= 1 cluster	Because **the first cluster address** is still readable, the recovery is having a very high successful rate. Note that filenames with **<u>the same postfix</u>** may also be found.

How to “*rescue*” a deleted file?

- If you’re really care about the deleted file, then…
  - **PULL THE POWER PLUG AT ONCE!**
  - Pulling the power plug stops the target clusters from being over-written.

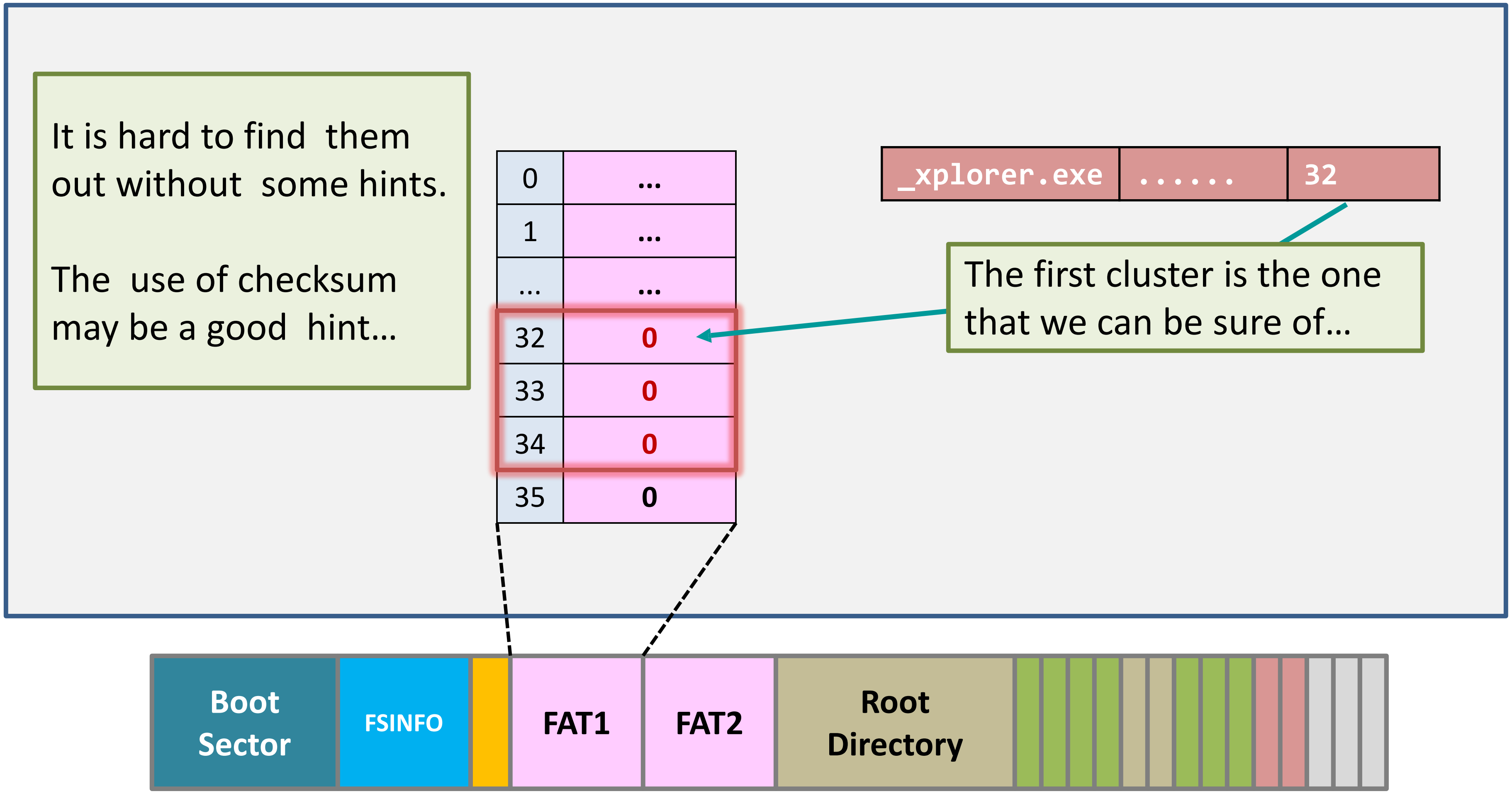
Principle of “rescue” deleted file

Data persists unless the sectors are reallocated and overwritten.

File size > 1 cluster	It is still possible as the clusters of a file are likely to be contiguously allocated. The next-available search provides a hint in looking for deleted blocks. If not, you’d better have the **checksum** and **the exact file size** beforehand, so that you can use a ***brute-force method*** to recover the file.

How to "*rescue*" a deleted file?

- What if the value of the 32nd cluster is not 0?


It is hard to find them out without some hints.
The use of checksum may be a good hint…
0 …
1 …
… …
32 0
33 0
34 0
35 0
_xplorer.exe
……
32
The first cluster is the one that we can be sure of…
Boot Sector
FSINFO
FAT1
FAT2
Root Directory

FAT series – conclusion

- It is a “nice” file system:
  - Space efficient: 4 bytes overhead (FAT entry) per data cluster.
- Deletion problem:
  - This is a lazy yet fast implementation.
  - Need extra protection for deleted data.
- Deployment:
  - It is everywhere: SD cards, USB drives, disks…