

Lecture 15: File System

Yinqian Zhang @ 2022, Spring

copyright@Bo Tang

Recall: C Low level I/O

- ◆ Operations on File Descriptors – as OS object representing the state of a file
 - ◆ User has a “handle” on the descriptor

```
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>

int open (const char *filename, int flags [, mode_t mode])
int create (const char *filename, mode_t mode)
int close (int filedes)
```

Bit vector of:

- Access modes (Rd, Wr, ...)
- Open Flags (Create, ...)
- Operating modes (Appends, ...)

Bit vector of Permission Bits:

- User|Group|Other X R|W|X

http://www.gnu.org/software/libc/manual/html_node/Opening-and-Closing-Files.html

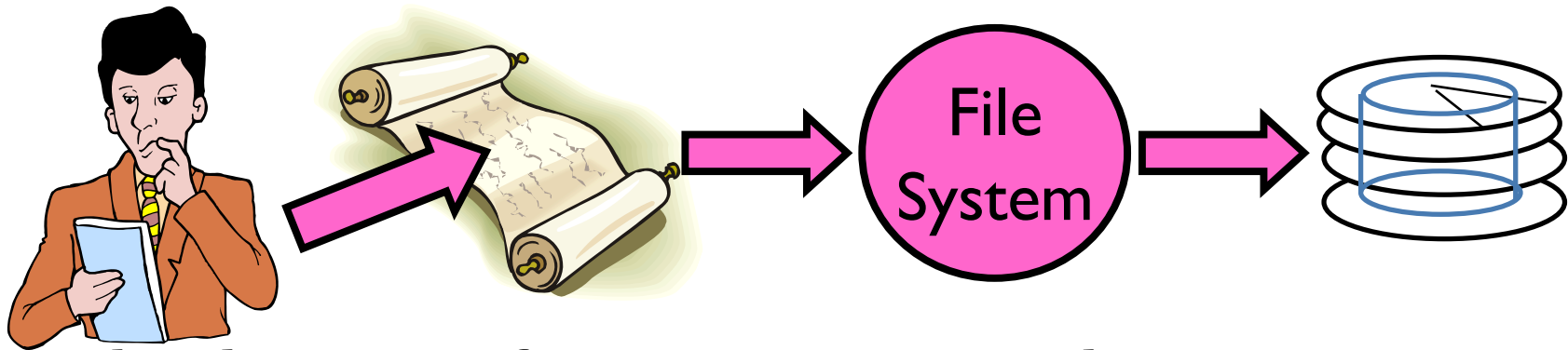
File System

- ◆ **File System:** Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.
- ◆ **File System Components**
 - ◆ **Naming:** Interface to find files by name, not by blocks
 - ◆ **Disk Management:** collecting disk blocks into files
 - ◆ **Protection:** Layers to keep data secure
 - ◆ **Reliability/Durability:** Keeping of files durable despite crashes, media failures, attacks, etc.

User vs. System View of a File

- ◆ User's view:
 - ◆ Durable Data Structures
- ◆ System's view (system call interface):
 - ◆ Collection of Bytes (UNIX)
 - ◆ Doesn't matter to system what kind of data structures you want to store on disk!
- ◆ System's view (inside OS):
 - ◆ Collection of blocks (a block is a logical transfer unit, while a sector is the physical transfer unit)
 - ◆ Block size \geq sector size; in UNIX, block size is 4KB

Translating from User to System View

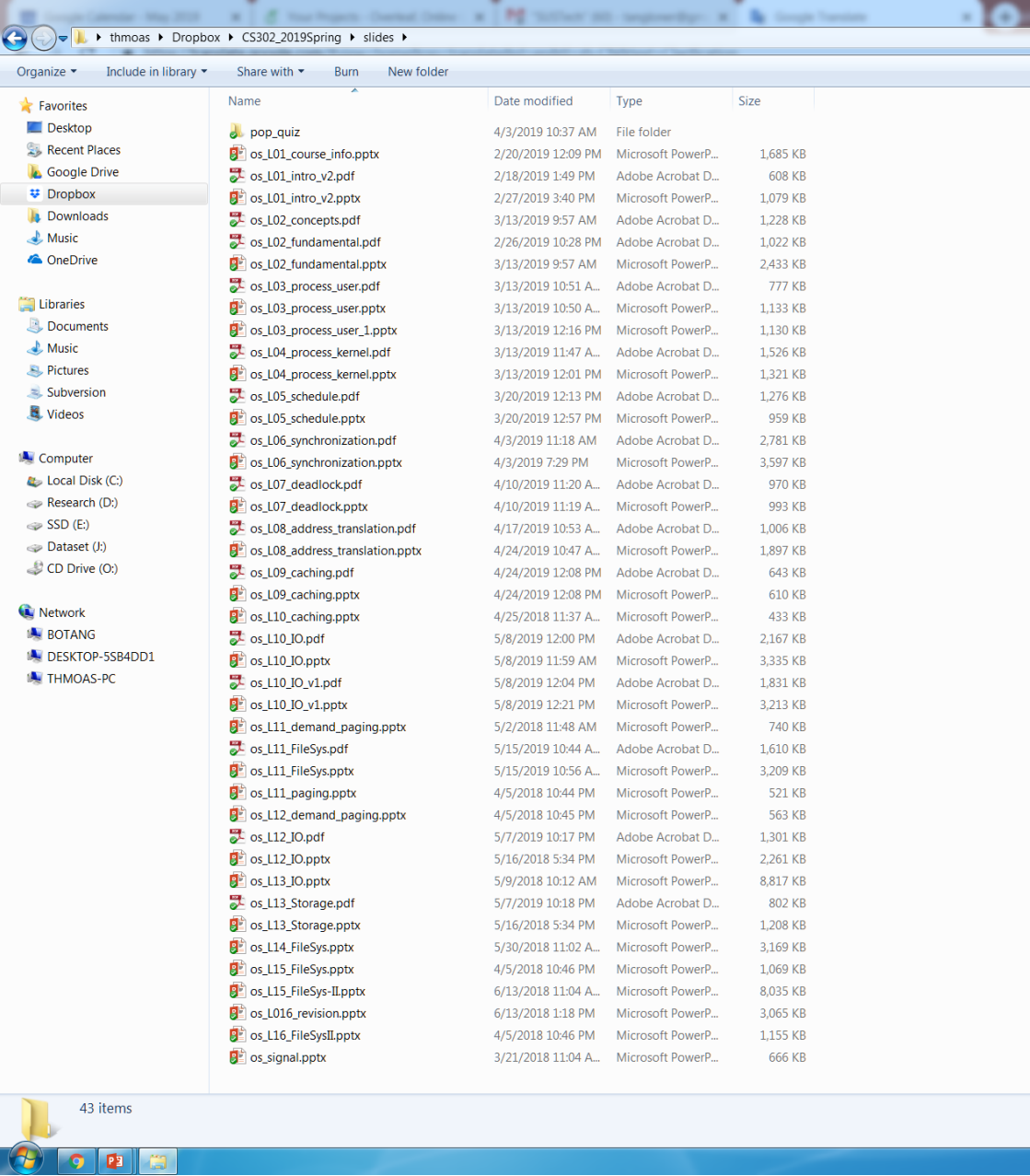


- ◆ What happens if user says: give me bytes 2—12?
 - ◆ Fetch block corresponding to those bytes
 - ◆ Return just the correct portion of the block
- ◆ What about: write bytes 2—12?
 - ◆ Fetch block
 - ◆ Modify portion
 - ◆ Write out Block
- ◆ Everything inside File System is in whole size blocks
 - ◆ For example, `getc()`, `putc()` \Rightarrow buffers something like 4096 bytes, even if interface is one byte at a time
- ◆ From now on, file is a collection of blocks

Directory

- ◆ Basically a hierarchical structure
- ◆ Each directory entry is a collection of
 - ◆ Files
 - ◆ Directories
 - ◆ A link to another entries
- ◆ Each has a name and attributes
 - ◆ Files have data
- ◆ Links (hard links) make it a DAG, not just a tree
 - ◆ Softlinks (aliases) are another name for an entry

Directories



The screenshot shows a Windows File Explorer window with the address bar displaying the path: `thmoas > Dropbox > CS302_2019Spring > slides`. The left sidebar shows the navigation pane with various locations, including Favorites, Libraries, Computer, and Network. The main pane displays a list of 43 items, including folders and files, with columns for Name, Date modified, Type, and Size.

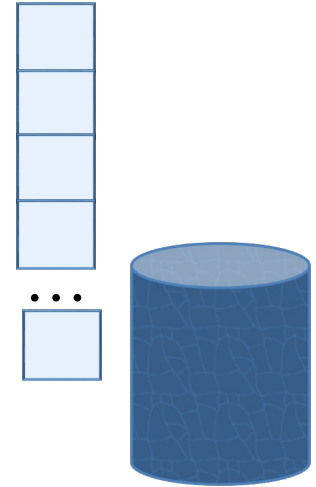
Name	Date modified	Type	Size
pop_quiz	4/3/2019 10:37 AM	File folder	
os_l01_course_info.pptx	2/20/2019 12:09 PM	Microsoft PowerP...	1,685 KB
os_l01_intro_v2.pdf	2/18/2019 1:49 PM	Adobe Acrobat D...	608 KB
os_l01_intro_v2.pptx	2/27/2019 3:40 PM	Microsoft PowerP...	1,079 KB
os_l02_concepts.pdf	3/13/2019 9:57 AM	Adobe Acrobat D...	1,228 KB
os_l02_fundamental.pdf	2/26/2019 10:28 PM	Adobe Acrobat D...	1,022 KB
os_l02_fundamental.pptx	3/13/2019 9:57 AM	Microsoft PowerP...	2,433 KB
os_l03_process_user.pdf	3/13/2019 10:51 A...	Adobe Acrobat D...	777 KB
os_l03_process_user.pptx	3/13/2019 10:50 A...	Microsoft PowerP...	1,133 KB
os_l03_process_user_1.pptx	3/13/2019 12:16 PM	Microsoft PowerP...	1,130 KB
os_l04_process_kernel.pdf	3/13/2019 11:47 A...	Adobe Acrobat D...	1,526 KB
os_l04_process_kernel.pptx	3/13/2019 12:01 PM	Microsoft PowerP...	1,321 KB
os_l05_schedule.pdf	3/20/2019 12:13 PM	Adobe Acrobat D...	1,276 KB
os_l05_schedule.pptx	3/20/2019 12:57 PM	Microsoft PowerP...	959 KB
os_l06_synchronization.pdf	4/3/2019 11:18 AM	Adobe Acrobat D...	2,781 KB
os_l06_synchronization.pptx	4/3/2019 7:29 PM	Microsoft PowerP...	3,597 KB
os_l07_deadlock.pdf	4/10/2019 11:20 A...	Adobe Acrobat D...	970 KB
os_l07_deadlock.pptx	4/10/2019 11:19 A...	Microsoft PowerP...	993 KB
os_l08_address_translation.pdf	4/17/2019 10:53 A...	Adobe Acrobat D...	1,006 KB
os_l08_address_translation.pptx	4/24/2019 10:47 A...	Microsoft PowerP...	1,897 KB
os_l09_caching.pdf	4/24/2019 12:08 PM	Adobe Acrobat D...	643 KB
os_l09_caching.pptx	4/24/2019 12:08 PM	Microsoft PowerP...	610 KB
os_l10_caching.pptx	4/25/2018 11:37 A...	Microsoft PowerP...	433 KB
os_l10_IO.pdf	5/8/2019 12:00 PM	Adobe Acrobat D...	2,167 KB
os_l10_IO.pptx	5/8/2019 11:59 AM	Microsoft PowerP...	3,335 KB
os_l10_IO_v1.pdf	5/8/2019 12:04 PM	Adobe Acrobat D...	1,831 KB
os_l10_IO_v1.pptx	5/8/2019 12:21 PM	Microsoft PowerP...	3,213 KB
os_l11_demand_paging.pptx	5/2/2018 11:48 AM	Microsoft PowerP...	740 KB
os_l11_FileSys.pdf	5/15/2019 10:44 A...	Adobe Acrobat D...	1,610 KB
os_l11_FileSys.pptx	5/15/2019 10:56 A...	Microsoft PowerP...	3,209 KB
os_l11_paging.pptx	4/5/2018 10:44 PM	Microsoft PowerP...	521 KB
os_l12_demand_paging.pptx	4/5/2018 10:45 PM	Microsoft PowerP...	563 KB
os_l12_IO.pdf	5/7/2019 10:17 PM	Adobe Acrobat D...	1,301 KB
os_l12_IO.pptx	5/16/2018 5:34 PM	Microsoft PowerP...	2,261 KB
os_l13_IO.pptx	5/9/2018 10:12 AM	Microsoft PowerP...	8,817 KB
os_l13_Storage.pdf	5/7/2019 10:18 PM	Adobe Acrobat D...	802 KB
os_l13_Storage.pptx	5/16/2018 5:34 PM	Microsoft PowerP...	1,208 KB
os_l14_FileSys.pptx	5/30/2018 11:02 A...	Microsoft PowerP...	3,169 KB
os_l15_FileSys.pptx	4/5/2018 10:46 PM	Microsoft PowerP...	1,069 KB
os_l15_FileSys-ll.pptx	6/13/2018 11:04 A...	Microsoft PowerP...	8,035 KB
os_l016_revision.pptx	6/13/2018 1:18 PM	Microsoft PowerP...	3,065 KB
os_l16_FileSysll.pptx	4/5/2018 10:46 PM	Microsoft PowerP...	1,155 KB
os_signal.pptx	3/21/2018 11:04 A...	Microsoft PowerP...	666 KB

43 items

File

- ◆ Named permanent storage
- ◆ Contains
 - ◆ Data
 - ◆ Blocks on disk somewhere
 - ◆ Metadata (Attributes)
 - ◆ Owner, size, last opened, ...
 - ◆ Access rights
 - ◇ R, W, X
 - ◇ Owner, Group, Other (in Unix systems)
 - ◇ Access control list in Windows system

Data blocks



Disk Management Policies (1/2)

- ◆ Basic entities on a disk:
 - ◆ **File**: user-visible group of blocks arranged sequentially in logical space
 - ◆ **Directory**: user-visible index mapping names to files
- ◆ Access disk as linear array of sectors.
 - ◆ Two Options:
 - ◆ Identify sectors as vectors [cylinder, surface, sector], sort in cylinder-major order, not used anymore
 - ◆ **Logical Block Addressing (LBA)**: Every sector has integer address from zero up to max number of sectors
 - ◆ Controller translates from address \Rightarrow physical position
 - ◆ First case: OS/BIOS must deal with bad sectors
 - ◆ Second case: hardware shields OS from structure of disk

Disk Management Policies (2/2)

- ◆ Need way to track free disk blocks
 - ◆ Link free blocks together \Rightarrow too slow today
 - ◆ Use bitmap to represent free space on disk
- ◆ Need way to structure files: **File Header**
 - ◆ Track which blocks belong at which offsets within the logical file structure
 - ◆ **Optimize placement of files' disk blocks to match access and usage patterns**

File System

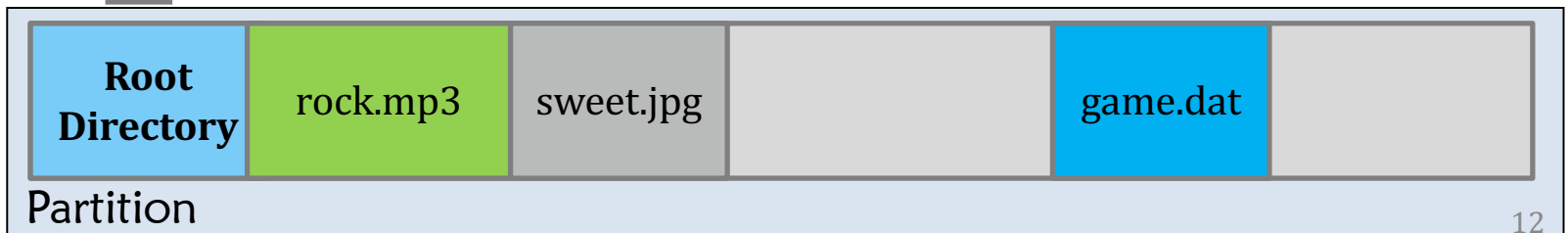
- ◆ Layout
 - ◆ contiguous allocation
 - ◆ linked allocation
 - ◆ inode allocation (next lecture)

Contiguous allocation – basics

Locate files easily.

Filename	Starting Address	Size
rock.mp3	100	1900
sweet.jpg	2001	1234
game.dat	5000	1000

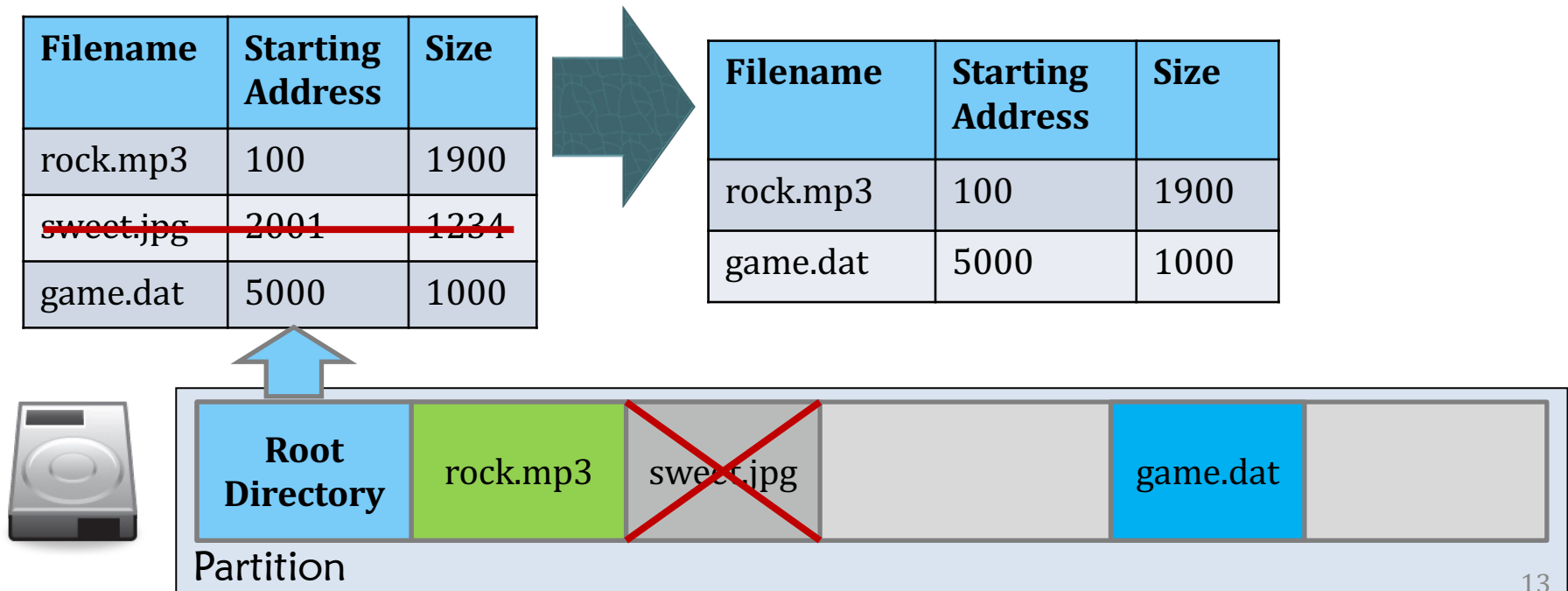
Free space is here



Contiguous allocation – basics

File deletion is easy! Space de-allocation is the same as updating the root directory!

Yet, how about file creation?

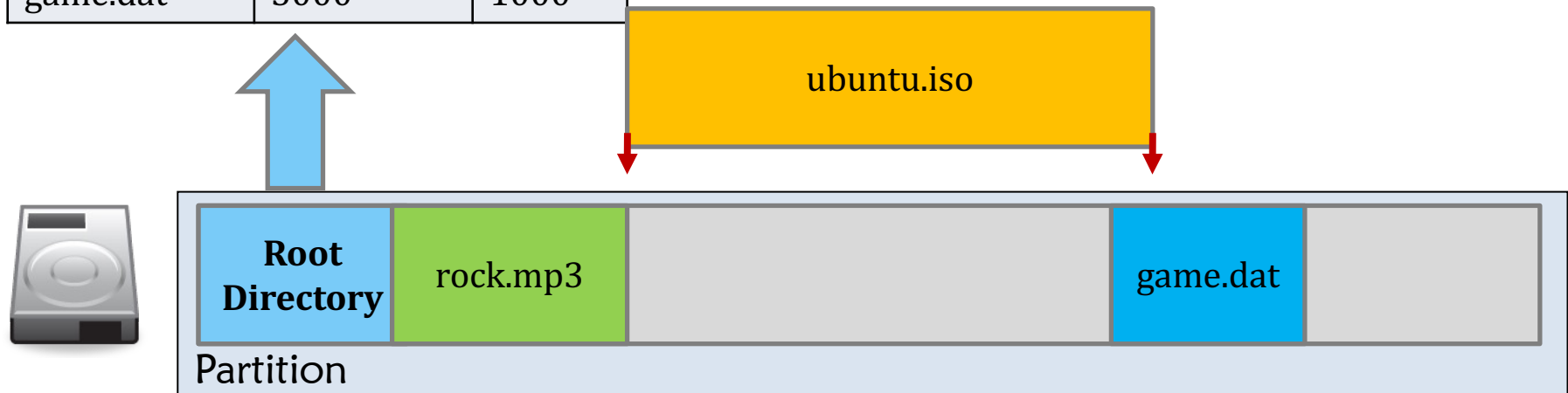


Contiguous allocation – basics

Really BAD! We have enough space, but there is no holes that I can satisfy the request. The name of the problem is called:

External Fragmentation

Filename	Starting Address	Size
rock.mp3	100	1900
game.dat	5000	1000

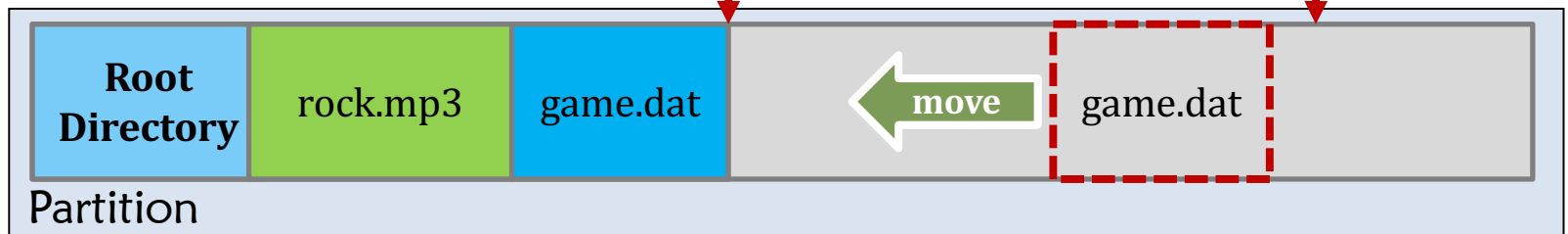


Contiguous allocation – basics

Defragmentation process may help!

You know, this is very expensive as you're working on disks.

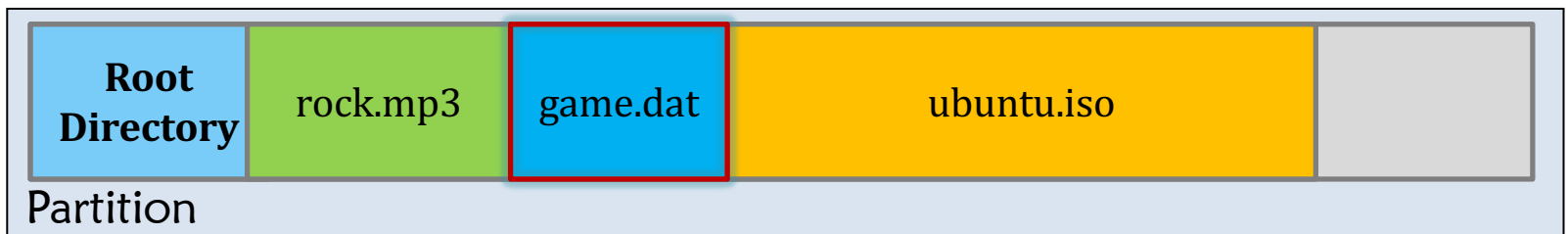
Filename	Starting Address	Size
rock.mp3	100	1900
game.dat	2001	1000
ubuntu...	3001	9000



Contiguous allocation – basics

Filename	Starting Address	Size
rock.mp3	100	1900
game.dat	2001	1000
ubuntu...	3001	9000

Growth problem!



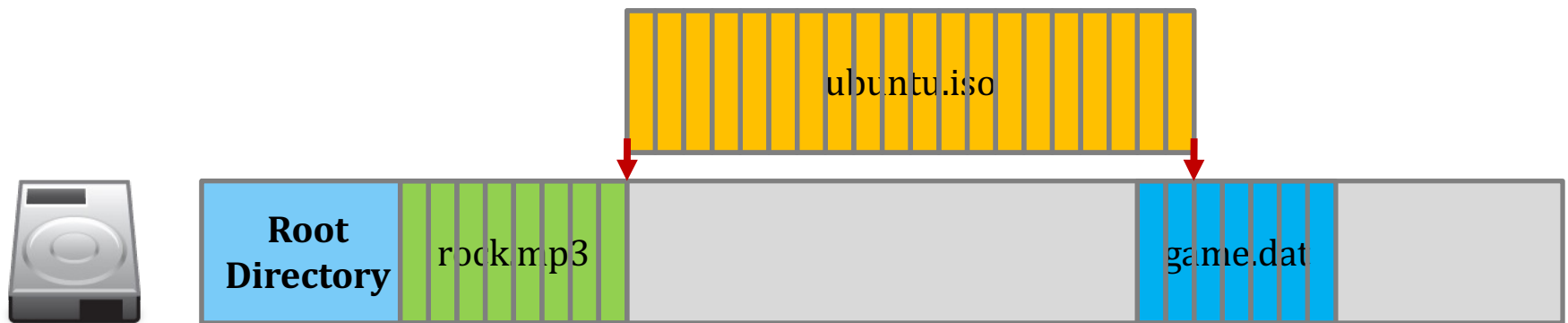
Contiguous allocation – application?

- ◆ ISO 9660
- ◆ CD-ROM
 - ◆ .iso image



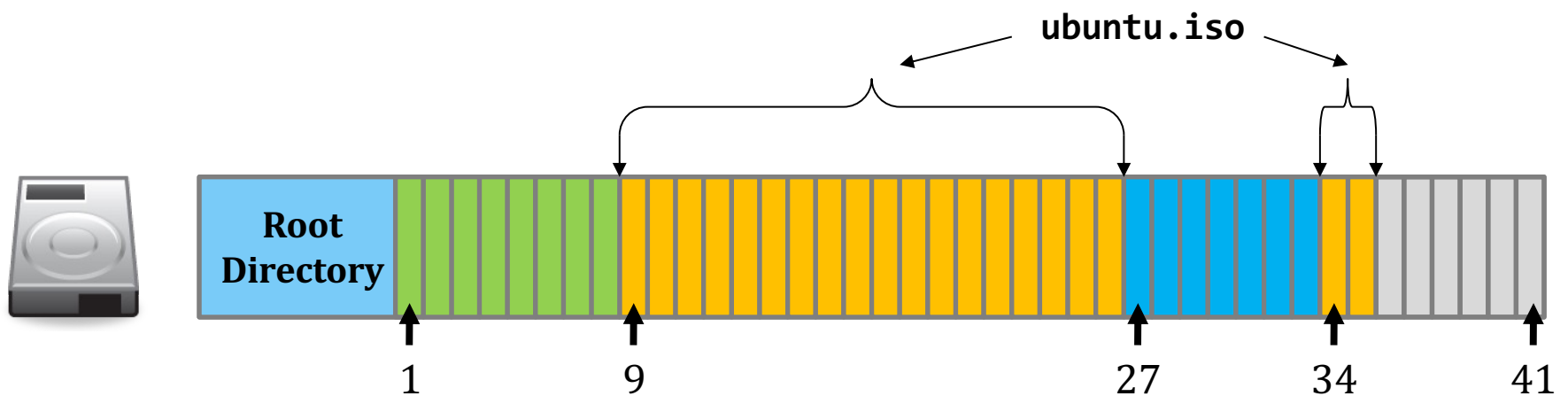
Linked allocation

- ◆ Let's borrow the idea from linked list...
 - ◆ Step (1) Chop the storage device and data into **equal-sized blocks**.



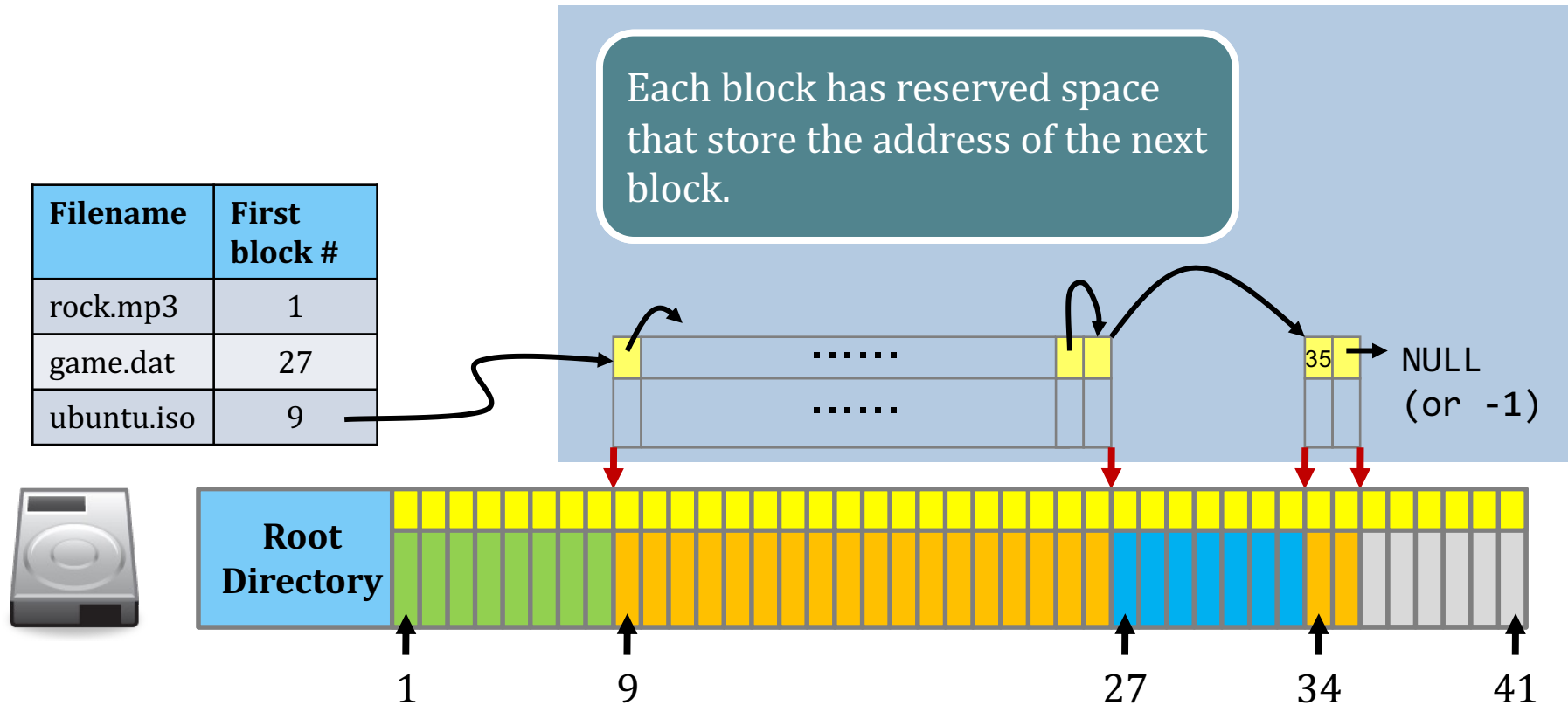
Linked allocation

- ◆ Let's borrow the idea from the linked list ...
 - ◆ Step (1) Chop the storage device into **equal-sized blocks**.
 - ◆ Step (2) Fill the empty space in a **block-by-block** manner.



Linked allocation

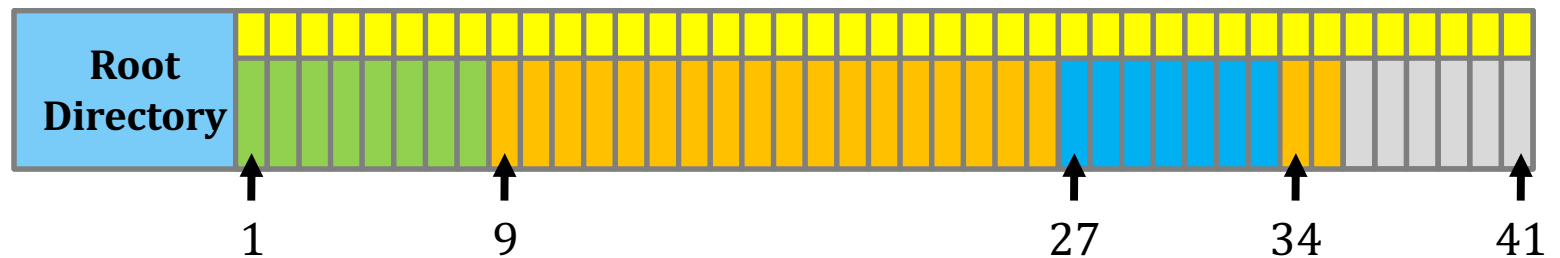
- ◆ Leave **4 bytes from each block** as the “pointer”
 - ◆ To write the block # of the next block into the first 4 bytes of each block.



Linked allocation

- ◆ Also keep the file size in the root directory table
 - ◆ To facilitate “ls -l” that lists the file size of each file
 - ◆ (otherwise needs to live counting how many blocks each file has)

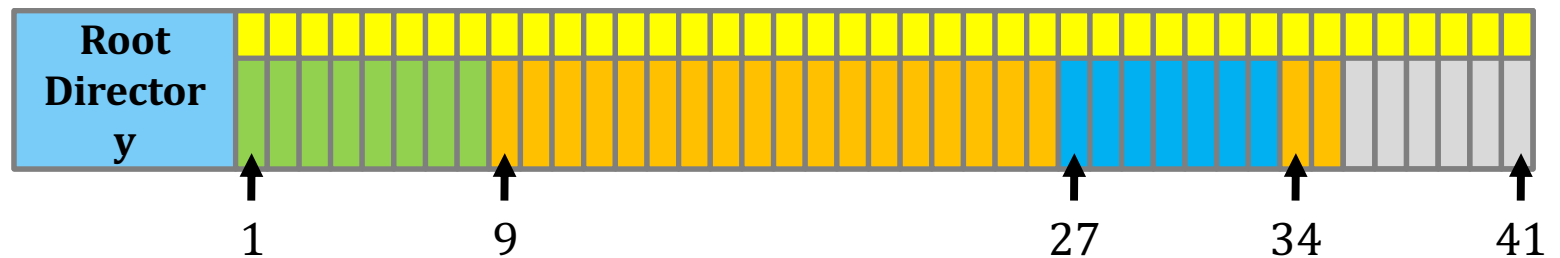
Filename	First block #	Size
rock.mp3	1	1900
game.dat	27	1000
ubuntu.iso	9	9000



Linked allocation

- ◆ So, how would you grade this file system?
 - ◆ External fragmentation?
 - ◆ File growth?

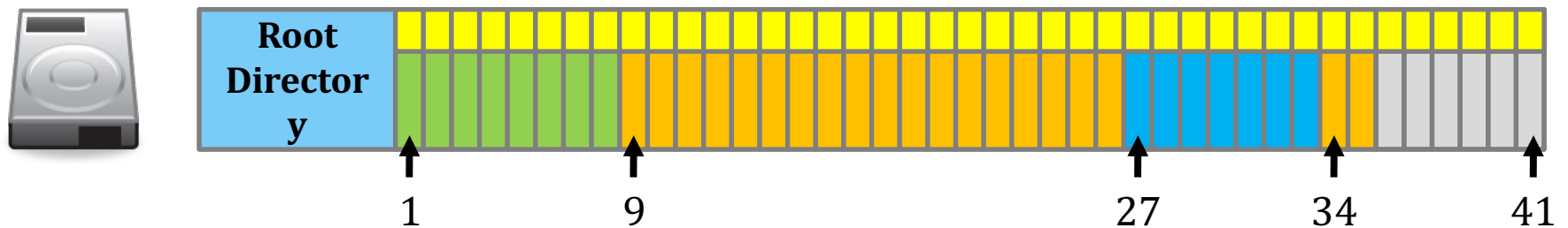
Filename	First block #	Size
rock.mp3	1	1900
game.dat	27	1000
ubuntu.iso	9	9000



Linked allocation

◆ Internal Fragmentation.

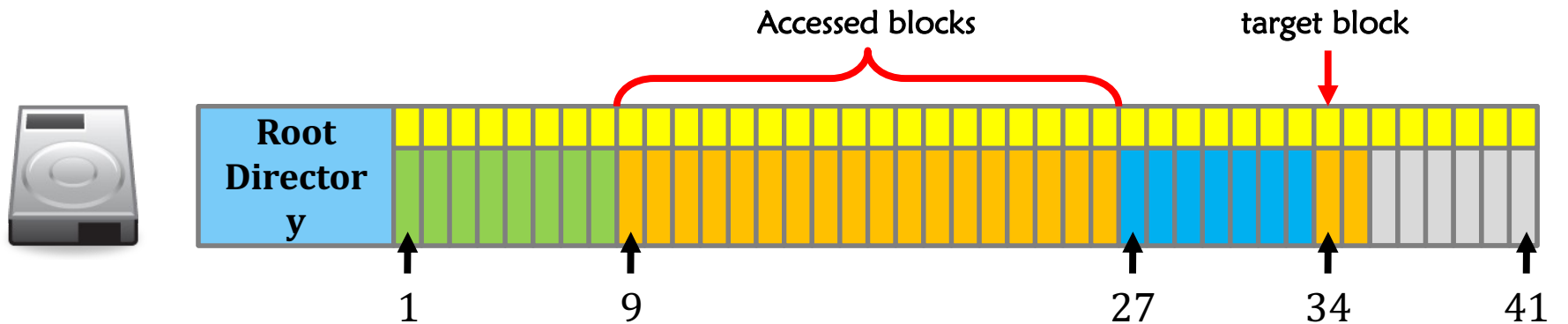
- ◆ A file is not always a multiple of the block size.
 - ◆ The last block of a file may not be **fully filled**.
 - ◆ E.g., a file of size 1 byte still occupies one block.
- ◆ The remaining space will be wasted since no other files can be allowed to fill such space.



Linked allocation

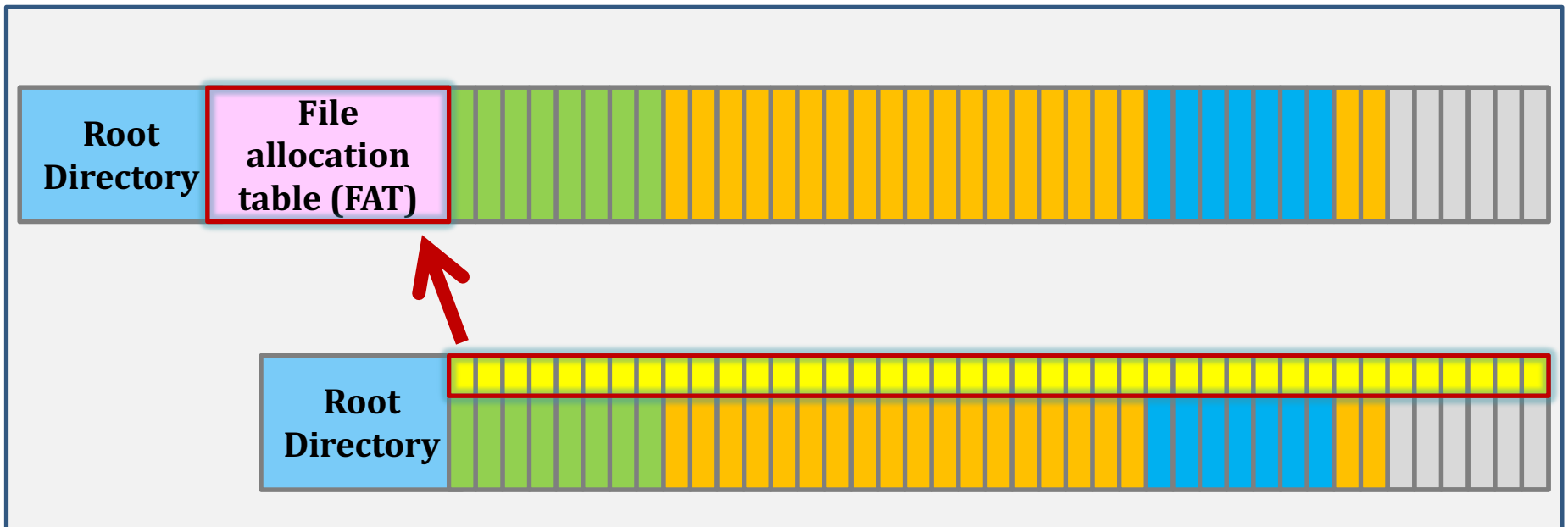
❖ **Poor random access performance.**

- ❖ What if I want to access the 2019-th block of ubuntu.iso?
- ❖ **You have to access blocks 1 – 2018 of ubuntu.iso until the 2019-th block**



FAT

- ❖ Centralize all the block links as File Allocation Table



FAT

Task: read “ubuntu.iso” sequentially.

Filename	First block #	Size
rock.mp3	1	1900
game.dat	27	1000
ubuntu.iso	9	9000

Step 1. Read the root directory and retrieve the **first block number**.

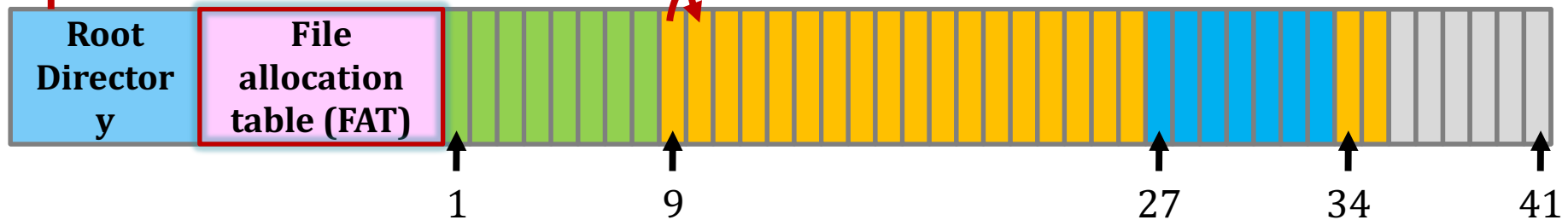
Step 2. Read the FAT to determine the location of next block.

File allocation table (FAT)

Block #	1	...	7	8	9	...	26	27	...	32	33	34	35	...	41
Next Block #	2	...	8	-1	10	...	34	28	...	33	-1	35	-1	...	0

1

2



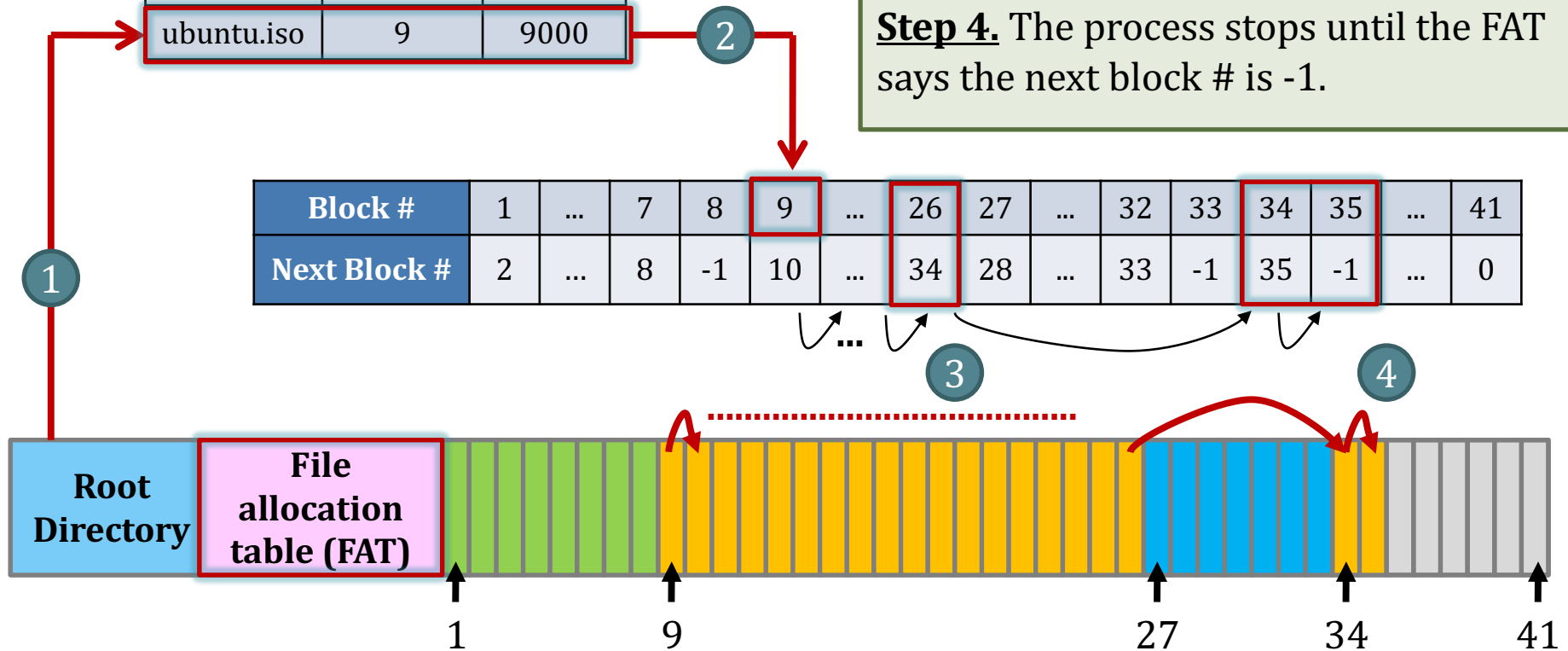
FAT

Task: read “ubuntu.iso” sequentially.

Filename	First block #	Size
rock.mp3	1	1900
game.dat	27	1000
ubuntu.iso	9	9000

Step 3. After reading the 2nd block, the process continues. Note that the blocks **may not be contiguously allocated**.

Step 4. The process stops until the FAT says the next block # is -1.

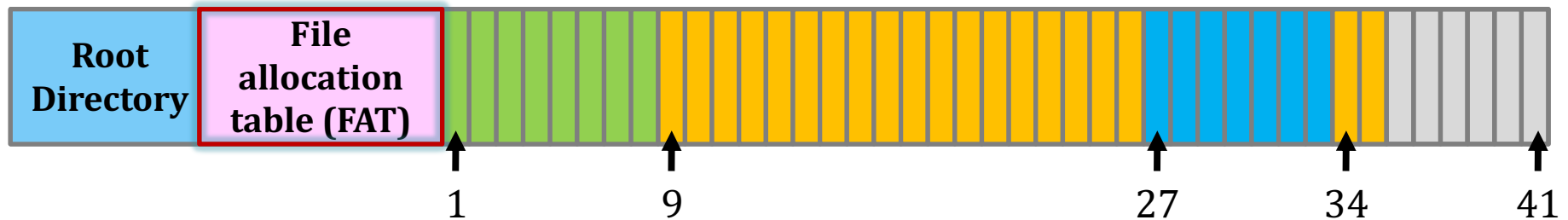


FAT

Resulting layout & file allocation.

Filename	First block #	Size
rock.mp3	1	1900
game.dat	27	1000
ubuntu.iso	9	9000

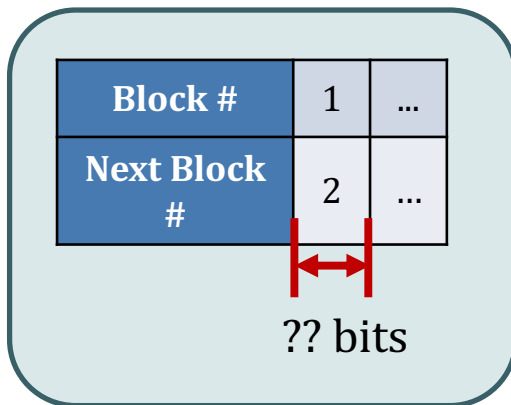
Block #	1	...	7	8	9	...	26	27	...	32	33	34	35	...	41
Next Block #	2	...	8	-1	10	...	34	28	...	33	-1	35	-1	...	0



FAT



- ◆ Start from floppy disk and DOS
- ◆ On DOS, a block is called as a '**cluster**'
- ◆ E.g., FAT12
 - ◆ 12-bit cluster address
 - ◆ Can point up to $2^{12} = 4096$ blocks



	FAT12	FAT16	FAT32
Cluster address length	12 bits	16 bits	28 bits
Number of clusters	2^{12} (4,096)	2^{16} (65,536)	2^{28}

MS reserves 4 bits (but nobody eventually used those)

FAT

◆ Size of a block (cluster):

Available block sizes (bytes)								
512	1K	2K	8K	16K	32K	64K	128K	256K

E.g.,

File system
size.

block size: 32KB

block address: 28 bits

$$\begin{aligned}(32 \times 2^{10}) \times 2^{28} &= 2^5 \times 2^{10} \times 2^{28} \\ &= 2^{43} \quad (8 \text{ TB})\end{aligned}$$

*** but MS deliberately set its formatting tool to format it up to 32GB only to lure you to use NTFS**

FAT series – layout overview

	Propose	Size
Reserved sectors	Boot sector	FS-specific parameters
	FSINFO	Free-space management
	More reserved sectors	Optional
	FAT (2 pieces)	1 copy as backup
	Root directory	Start of the directory tree.
		Variable, can be changed during formatting
		Variable, depends on disk size and cluster size.
		At least one cluster, depend on the number of directory entries.



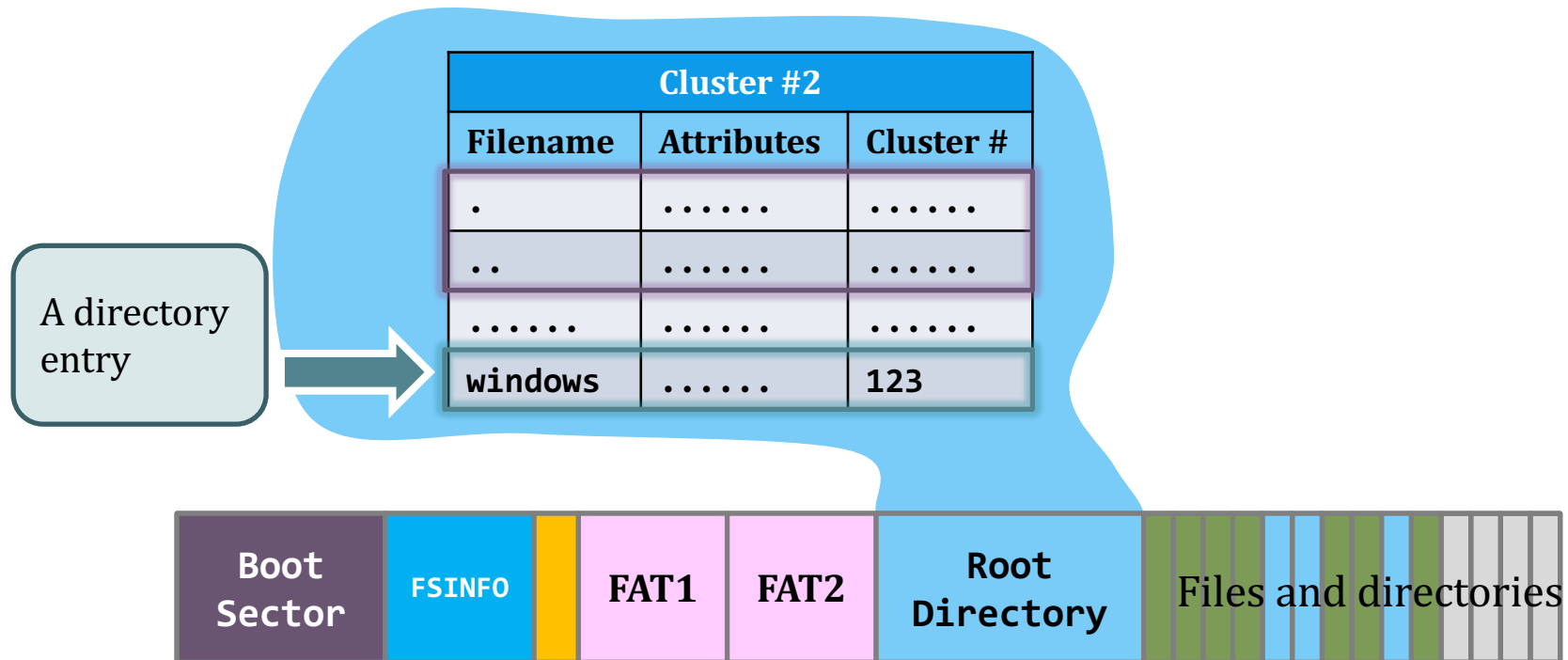
A FAT partition

FAT series – 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
.....
06/13/2007  1,033,216   gamedata.dat
08/04/2004    69,120   notepad.exe
.....
c:\> _
```

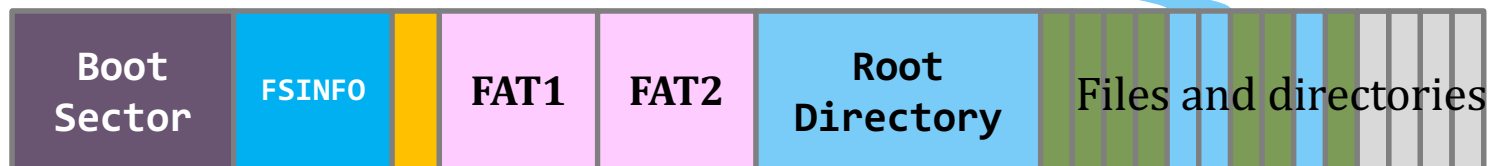


FAT series – directory traversal

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

```
c:\> dir c:\windows
.....
06/13/2007  1,033,216  gamedata.dat
08/04/2004   69,120  notepad.exe
.....
c:\> _
```

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



FAT series – directory entry

- ◆ A 32-byte directory entry in a directory file
- ◆ A directory entry is describing a file (or a sub-directory) under a particular directory

Bytes	Description
0-0	1 st character of the filename (0x00 or 0xe5 means unallocated)
1-10	remaining 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.

Filename	Attributes	Cluster #
explorer.dat	32

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

FAT series – directory entry

- ◆ The 1st block address of that file

Bytes	Description
0-0	1 st 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.

Filename	Attributes	Cluster #
explorer.dat	32

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

FAT series – directory entry

◆ Directory entry is just a structure.

Bytes	Description
0-0	1 st 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.

Filename	Attributes	Cluster #
explorer.dat	32

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 FAT32 file?

4G - 1 bytes

Bounded by the file size attribute!

Why “- 1”?

- Imagine 3 bits: 000, 001, ..., 110, 111
- Largest number is 111 = $2^3 - 1$
- i.e., we also need to represent “0 bytes”

FAT series – LFN directory entry

◆ LFN: Long File Name.

◆ In old days, Uncle Bill set the rule that every file should follow the 8+3 naming convention.

◆ To support LFN



- ◆ Abuse directory entries to store the file name!
- ◆ Allow to use up to 20 entries for one LFN

Directory file
LFN ...
LFN #2
LFN #1
Normal Entry

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

A normal directory entry is **still** there.

FAT series – LFN directory entry

❖ Normal directory entry vs LFN directory entry

Bytes	Description
0-0	1 st 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.

Bytes	Description
0-0	Sequence Number
1-10	File name characters (5 characters in Unicode)
11-11	File attributes - always 0x0F (to indicate it is a LFN)
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.

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 – 1 directory entry can hold

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	43	6d 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	02	65 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	01	49 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 cluster number.
 - ◆ It stores the **file size**; without the file size, how can you know when you should stop reading a cluster?
 - ◆ It stores **all file attributes**.

FAT series – reading a file

Task: read “C:\windows\gamedata.dat” sequentially.

FAT1

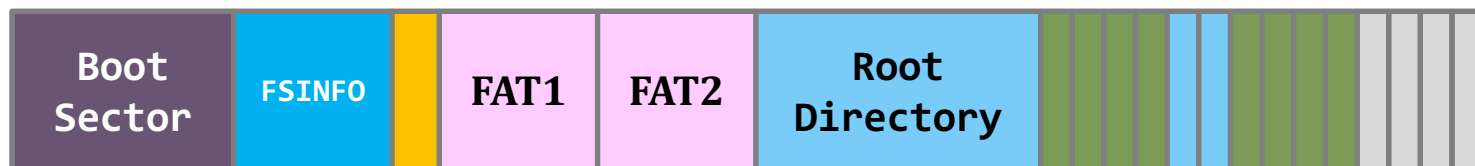
0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

Damaged	= 0xffffffff7
EOF	>= 0xffffffff8
Unallocated	= 0x0

Filename	Attributes	Cluster #
gamedata.dat	32

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

Step 2. Look for the next cluster and it is Cluster #33.



FAT series – reading a file

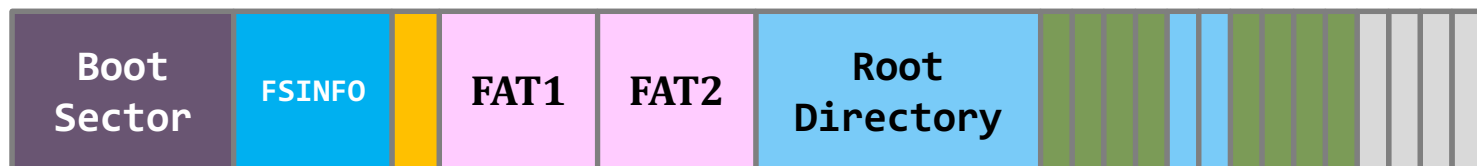
Task: read “C:\windows\gamedata.dat” sequentially.

0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

Filename	Attributes	Cluster #
gamedata.dat	32

Step 3. Since the FAT has marked “EOF”, we have reached the last cluster of that file.

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



FAT series – writing a file

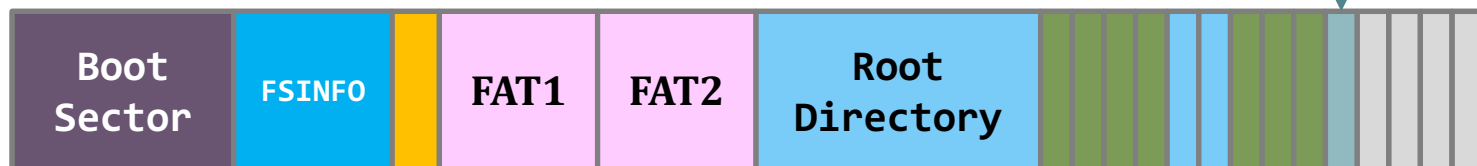
Task: append data to “C:\windows\gamedata.dat”.

0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

Filename	Attributes	Cluster #
gamedata.dat	32

Step 1. Locate the last cluster.

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



FAT series – writing a file

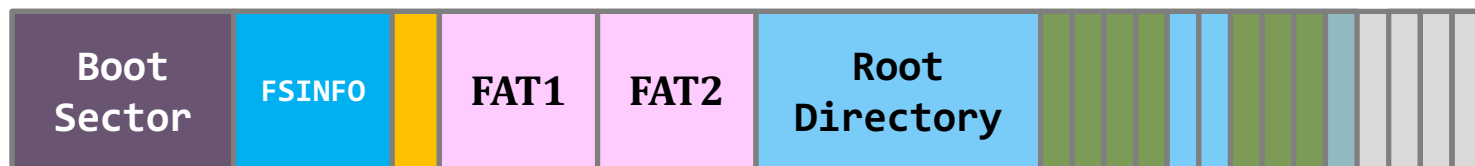
Task: append data to “C:\windows\gamedata.dat”.

0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

FSINFO	
# of free clusters	4
Next free cluster #	34

Filename	Attributes	Cluster #
gamedata.dat	32

Step 3. Allocate the next cluster through FSINFO.



FAT series – writing a file

Task: append data to “C:\windows\gamedata.dat”.

0	...
1	...
...	...
32	33
33	34
34	EOF
35	0

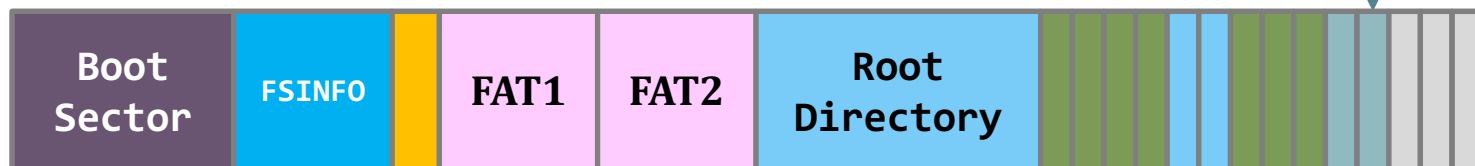
FSINFO	
# of free clusters	3
Next free cluster #	35

Filename	Attribute s	Cluster #
gamedata.dat	32

Step 3. Allocate the next cluster through FSINFO.

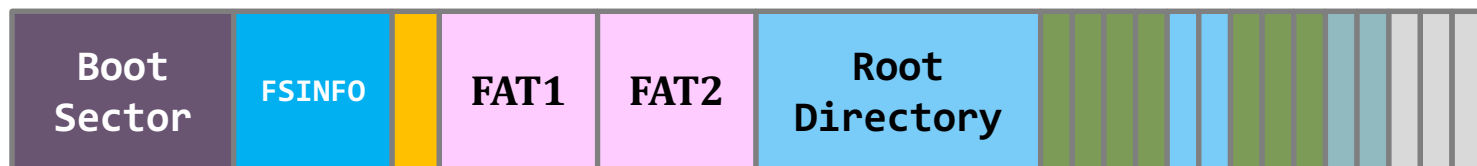
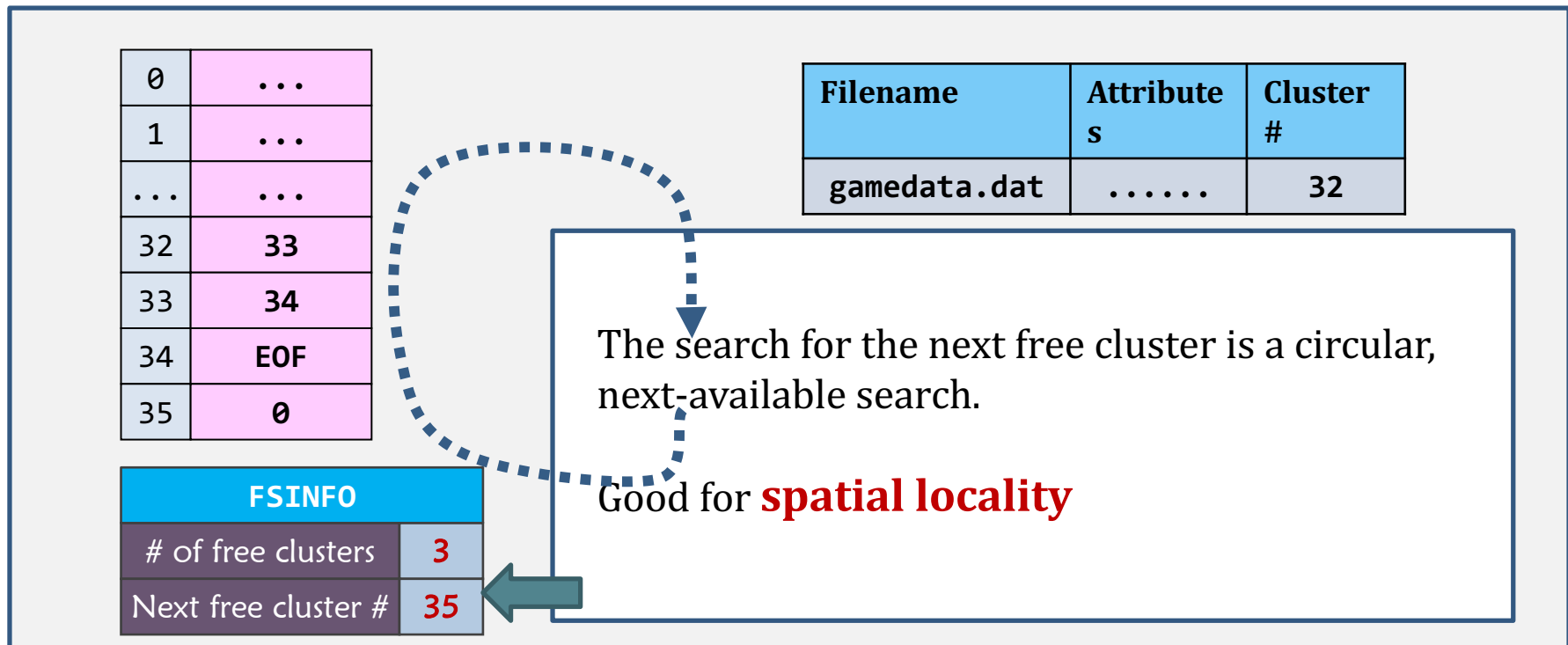
Step 4. Update the FATs and FSINFO.

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



FAT series – writing a file

Task: append data to “C:\windows\gamedata.dat”.



FAT series – delete a file

Task: delete “C:\windows\gamedata.dat”.

0	...
1	...
...	...
32	33
33	34
34	EOF
35	0



0	...
1	...
...	...
32	0
33	0
34	0
35	0

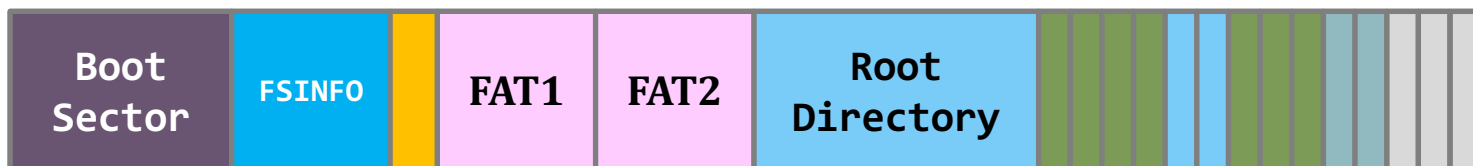
Filename	Attributes	Cluster #
gamedata.dat	32

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



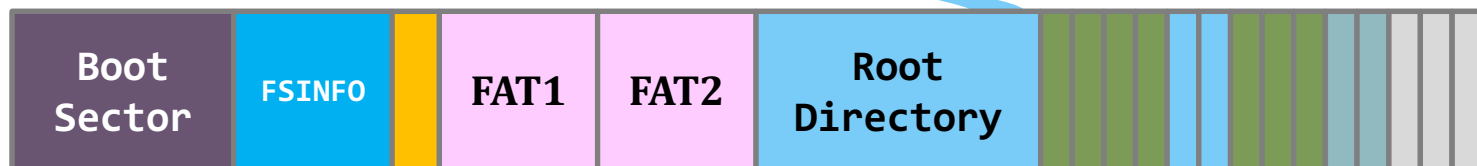
FAT series – delete a file

Task: delete “C:\windows\gamedata.dat”.

Directory “windows”		
Filename	Attribute s	Cluster #
.	?
..	?
_amedata.dat	32
notepad.exe	456

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

That's the end of deletion!



FAT series – 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**?

FAT series – how to recover a deleted file?

- ◆ If you 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.

File size is within one block (cluster)	Because the first cluster address in the direct is still readable, the recovery is having a very high successful rate.
File size spans more than 1 block	Because of the next-available search, clusters of a file are likely to be contiguous allocated. This provides a hint in looking for deleted blocks. Can you devise an undelete algorithm for FAT32?

FAT series – conclusion

- ◆ Space efficient:
 - ◆ 4 bytes overhead (FAT entry) per data cluster.
- ◆ Delete:
 - ◆ Lazy delete efficient
 - ◆ Insecure
 - ◆ designed for single-user 20+ years ago
- ◆ Deployment: (FAT32 and FAT12)
 - ◆ It is everywhere: CF cards, SD cards, USB drives
- ◆ Search:
 - ◆ Block addresses of a file may scatter discontinuously
 - ◆ To locate the 888-th block of a file?
 - ◆ Start from the first FAT entry and follow 888 pointers
- ◆ The most commonly used **filesystem** in the world

Designing a File System ...

- ◆ What factors are critical to the design choices?
- ◆ Durable data store => it's all on disk
- ◆ (Hard) Disks Performance !!!
 - ◆ Maximize sequential access, minimize seeks
- ◆ Open before Read/Write
 - ◆ Can perform protection checks and look up where the actual file resource are, in advance
- ◆ Size is determined as they are used !!!
 - ◆ Can write (or read zeros) to expand the file
 - ◆ Start small and grow, need to make room
- ◆ Organized into directories
 - ◆ What data structure (on disk) for that?
- ◆ Need to allocate / free blocks
 - ◆ Such that access remains efficient

Summary

- ◆ File System:
 - ◆ Transforms blocks into Files and Directories
 - ◆ Optimize for access and usage patterns
 - ◆ Maximize sequential access, allow efficient random access
- ◆ File Allocation Table (FAT) Scheme
 - ◆ Linked-list approach
 - ◆ Very widely used: Cameras, USB drives, SD cards
 - ◆ Simple to implement, but poor performance and no security

Thank You!