HCMUS - FIT

File System

Lecturer: VU THI MY HANG

OPERATING SYSTEM

Plan

- File System Interface
- File System Implementation
- Disk Storage

Plan

- **File System Interface**
- File System Implementation
- Disk Storage

File Concept

- An abstract concept
- A logical storage unit of a collection of related data on disks
- File types:
  - ✓ Ordinary/Regular file: ASCII or binary files
  - ✓ Directory file
  - ✓ Shortcut file
  - ✓ Special file (i.e., device file that represents a physical device)
- File attributes (i.e., metadata used for file management): name, size, location, creator, date (creation, access, modification), type, permission/protection, etc.

File Operations

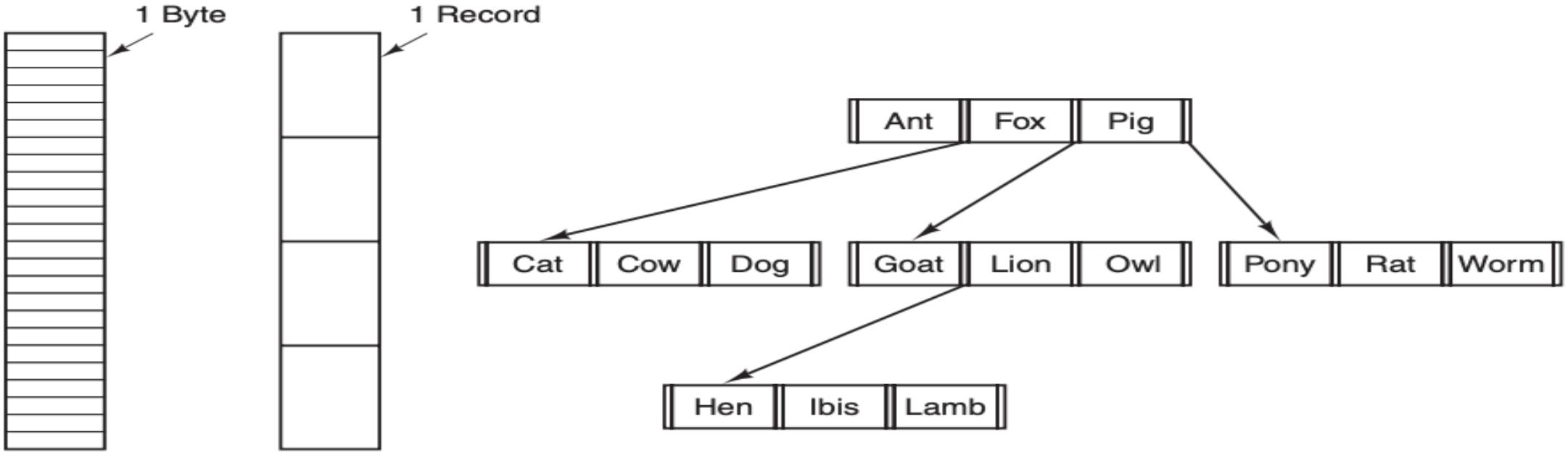
System call	Description
fd = creat(name, mode)	One way to create a new file
fd = open(file, how, ...)	Open a file for reading, writing or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information
s = fstat(fd, &buf)	Get a file's status information
s = pipe(&fd[0])	Create a pipe
s = fcntl(fd, cmd, ...)	File locking and other operations

Access methods

- Sequential access:
  - ✓ Data blocks are accessed one by one in order
- Direct random access:
  - ✓ A desired data block is accessed directly without traversing its prior blocks
- Indexed sequential access:
  - ✓ Assign an index to some data blocks for accessing these blocks
  - ✓ Built on top sequential access

File Structure

- Sequence of unstructured bytes/words (most used in modern OS, including Windows, Linux)
- Sequence of fixed-length records (used in early mainframes)
- Tree of records (still used in large systems for commercial processing)



Directory Concept

- Directory: a collection of files/subdirectories
- Two types of path for accessing a file/directory:
  - ✓ Absolute path: C:\user\home\...\myfile.doc
  - ✓ Relative path: .\myfile.doc
- Directory attributes: name (without extension), path, creator, date (creation, access, modification), permission/protection

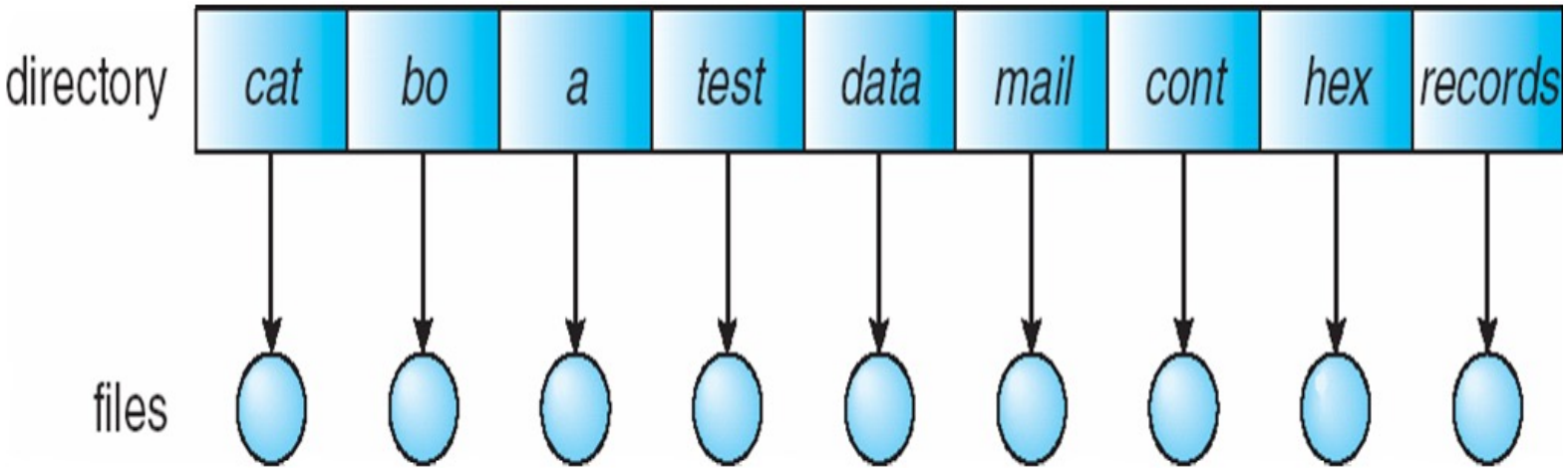
Directory Operations

System call	Description
s = mkdir(path, mode)	Create a new directory
s = rmdir(path)	Remove a directory
s = link(oldpath, newpath)	Create a link to an existing file
s = unlink(path)	Unlink a file
s = chdir(path)	Change the working directory
dir = opendir(path)	Open a directory for reading
s = closedir(dir)	Close a directory
dirent = readdir(dir)	Read one directory entry
rewinddir(dir)	Rewind a directory so it can be reread

Directory Organization

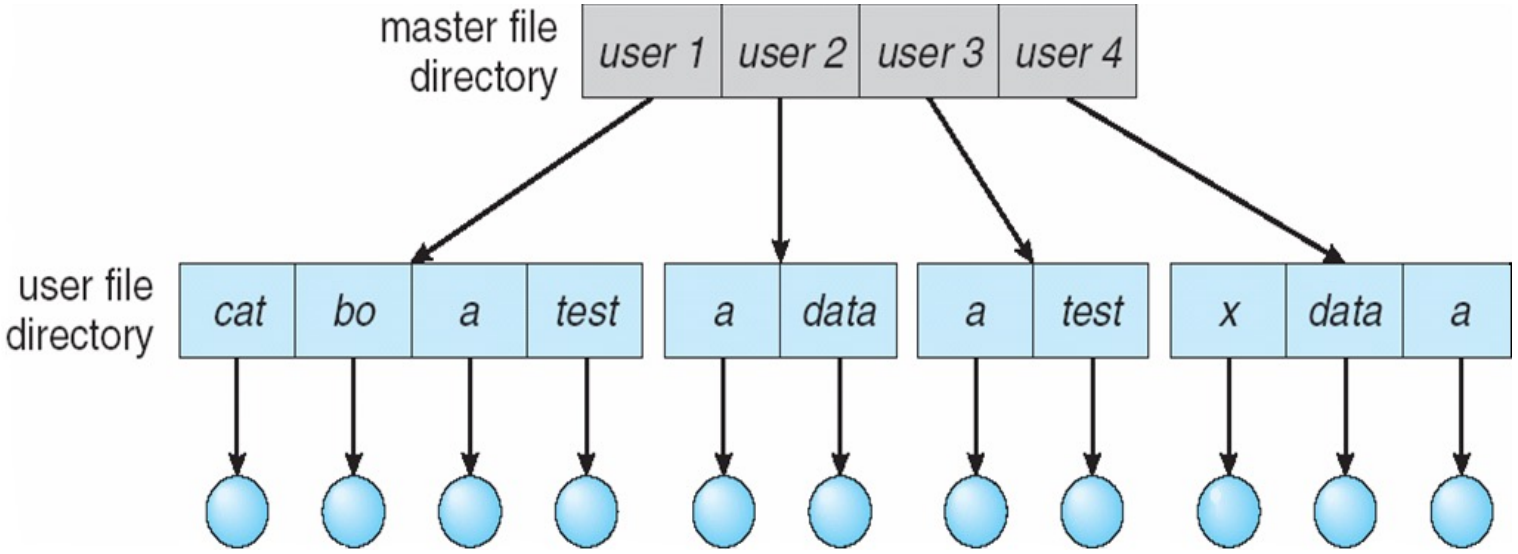
- Criteria
  - ✓ Efficiency (in searching/accessing a file)
  - ✓ Naming
  - ✓ Grouping capability according to users' need
  - ✓ Sharing
  - ✓ …

Directory Organization (cont.)



Single-level directory

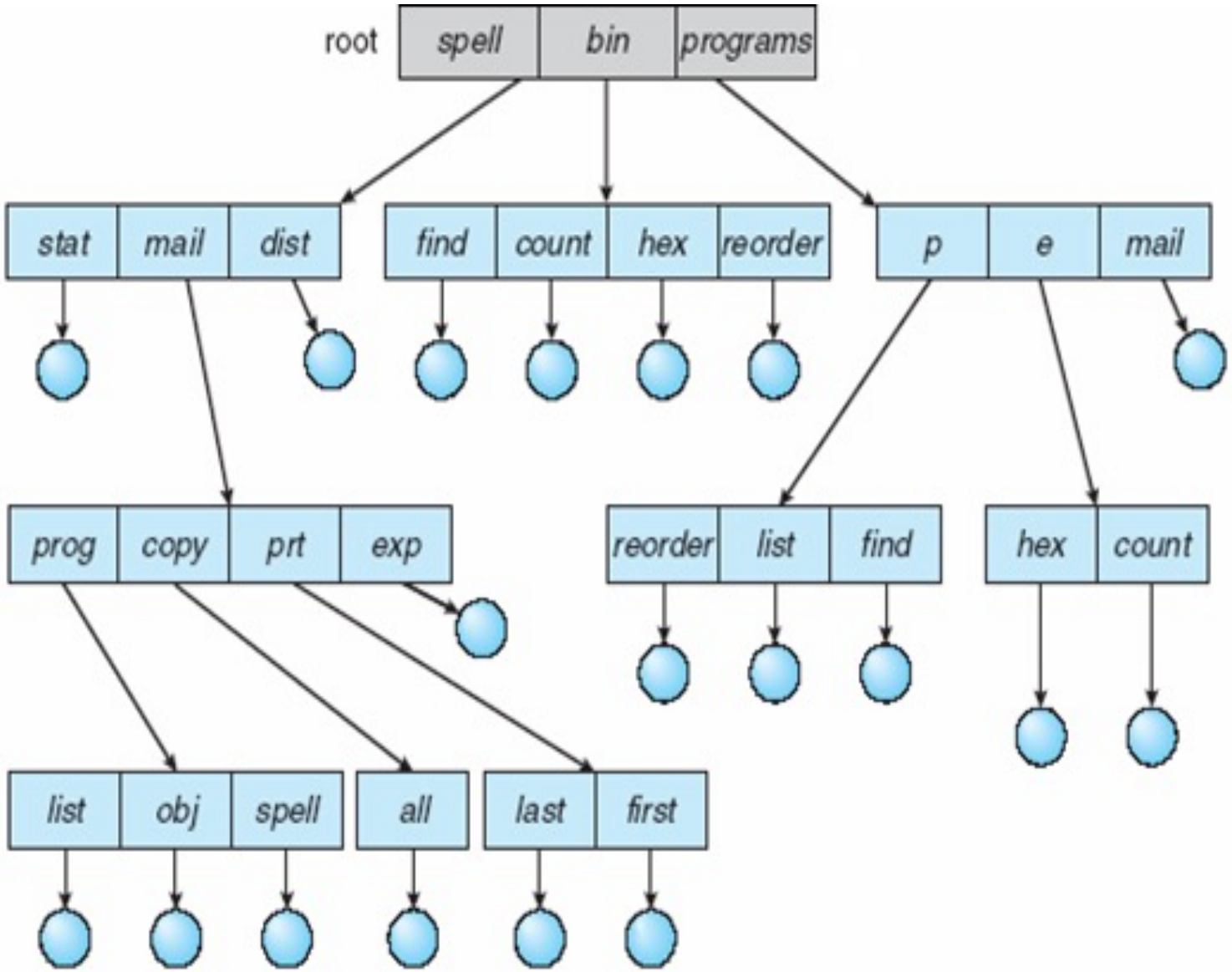
- ☺ Implementation
- ☺ Searching
- ☹ Naming
- ☹ Grouping



Two-level directory

- ☺ Naming
- ☺ Searching
- ☹ Grouping

Directory Organization (cont.)

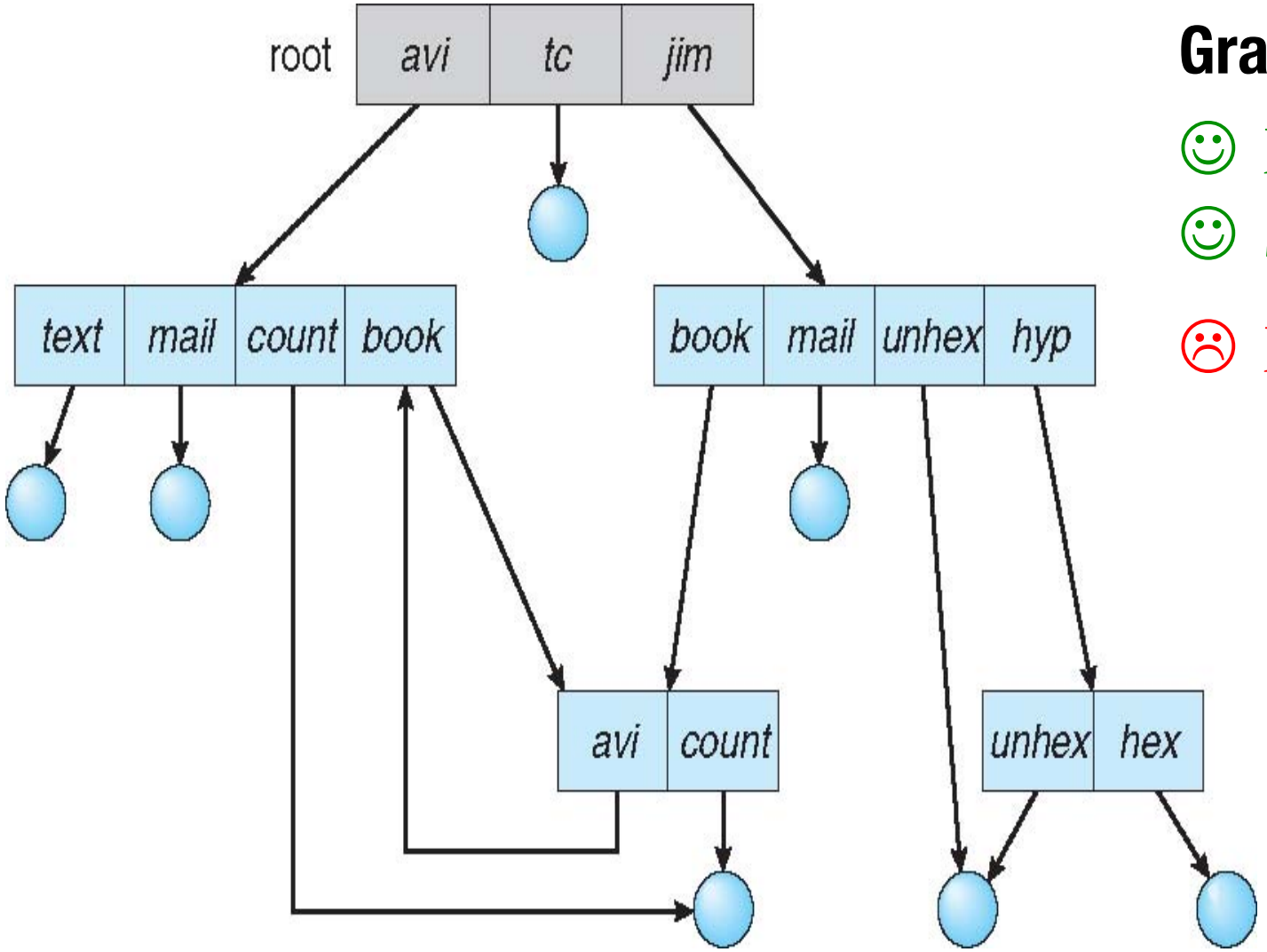


Tree-structure directory

☺ Naming
☺ Grouping
☺ Searching

☹ Accessing a file
☹ Same file in multiple directories
☹ Sharing

Directory Organization (cont.)



Graph directory

☺ More flexible

☺ Sharing

☹ More costly

Protection

- Mode of access: read (R), write(W), execute(X)
- Three classes of users: owner, group, universe (public)
- Example of access control list (ACL): rwxrw---x or 761

File permissions

Directory permissions

-rw-rw-r--	1 pbg	staff	31200	Sep 3 08:30	intro.ps
drwx------	5 pbg	staff	512	Jul 8 09.33	private/
drwxrwxr-x	2 pbg	staff	512	Jul 8 09:35	doc/
drwxrwx---	2 jwg	student	512	Aug 3 14:13	student-proj/
-rw-r--r--	1 pbg	staff	9423	Feb 24 2012	program.c
-rwxr-xr-x	1 pbg	staff	20471	Feb 24 2012	program
drwx--x--x	4 tag	faculty	512	Jul 31 10:31	lib/
drwx------	3 pbg	staff	1024	Aug 29 06:52	mail/
drwxrwxrwx	3 pbg	staff	512	Jul 8 09:35	test/

Example of Unix permissions

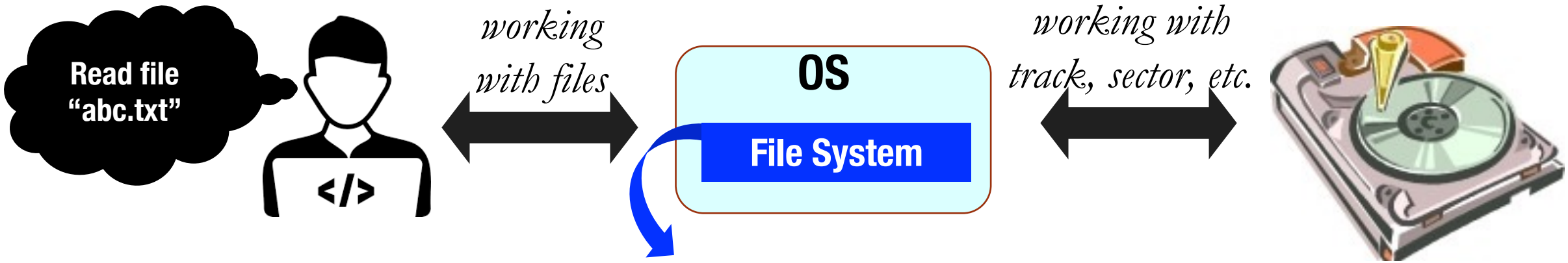
			RWX
a) owner access	7	⇒	1 1 1
b) group access	6	⇒	1 1 0
c) universe access	1	⇒	0 0 1

Plan

- File System Interface
- **File System Implementation**
  - ***Overview***
  - *File and Directory Implementation*
  - *File System Structure*
- Disk Storage

File System

- User's point of view: how a file is named, structured, manipulated
- System's point of view: how to store a file on disk blocks



A part of OS dealing with files:

- *Provide users an easy way to access files without awareness of the physical structure of storage devices*
- *Manage and map files onto disk drives (e.g., allocation/deallocation, read/write, protection, sharing)*

Secondary Storage: Disk Drive

- A hard disk drive can be divided into multiple partitions
- Need to install a file system on each partition to manipulate files storing on it
- Examples of common file systems
  - ✓ NTFS (Windows)
  - ✓ ext2, ext3, ext4 (Linux)
  - ✓ HFS, HFS+, APFS (macOS)
  - ✓ FAT32, exFAT (flash disk, removable device)

Windows File System: FAT/exFAT

- File Allocation Table (FAT): early file system on Windows
- Variants: FAT12/FAT16/FAT32
  - ✓ FAT32 is compatible with most of OS (other versions of Windows, macOS, Linux, and many others)

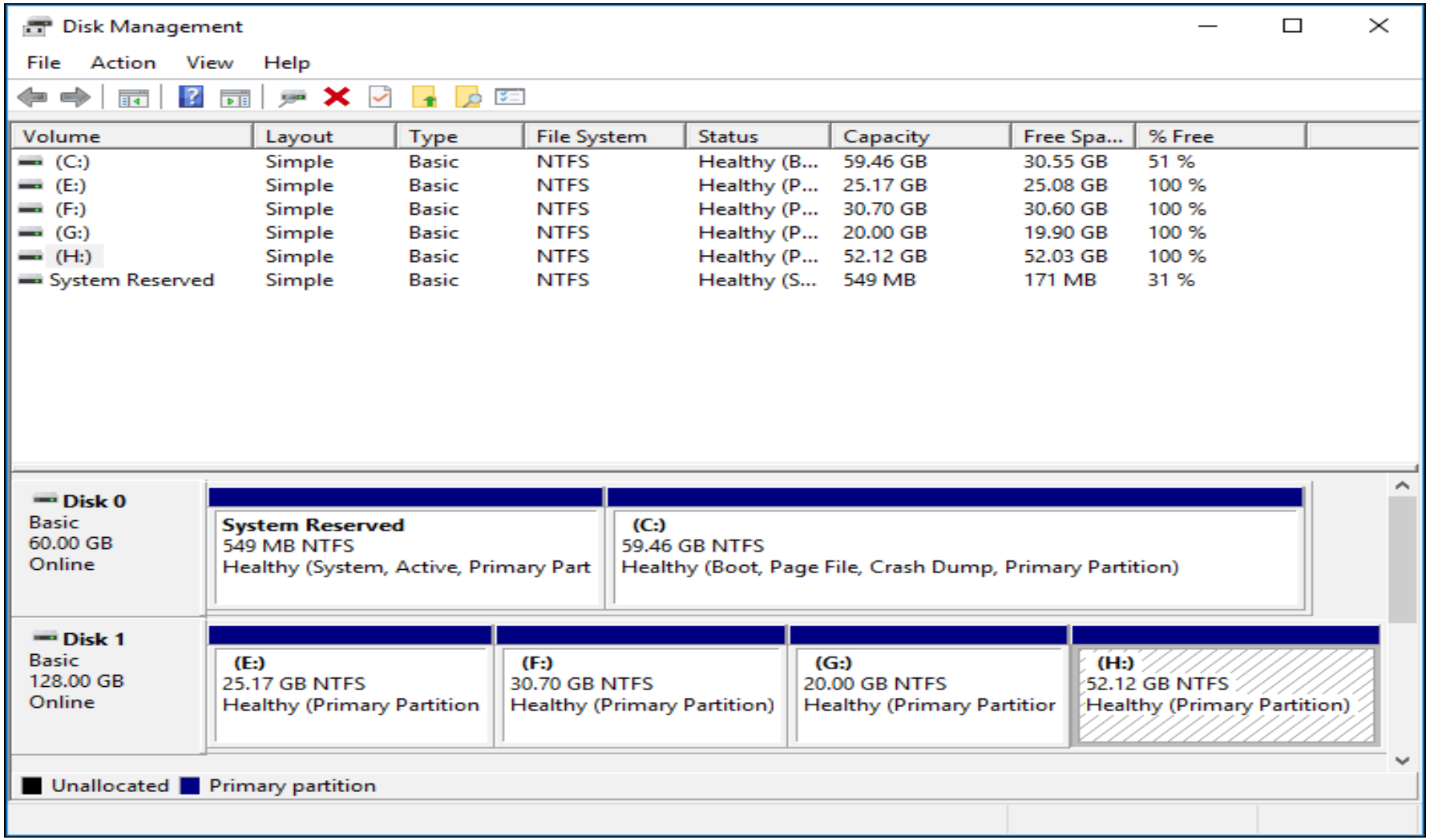
    → *still used for USB or removable devices (no files larger than 4GB)*
- Enhanced version of FAT: exFAT (extended FAT)
  - ✓ exFAT is compatible with all other versions of Windows, modern macOS, and Linux with some additional software required

    → *appropriate choice for flash drives (maximum file size 16EB)*

Windows File System: NTFS

- New Technology File System (NTFS)
- Primary file system for recent versions of Windows
- Advanced features: journaling, backup & restore, file and folder protection, compression, decryption/encryption, ...
- Support for large files and volumes
  - ✓ Maximum file size: 16EB
- Limited support in macOS (read-only support) and Linux

Windows File System: NTFS

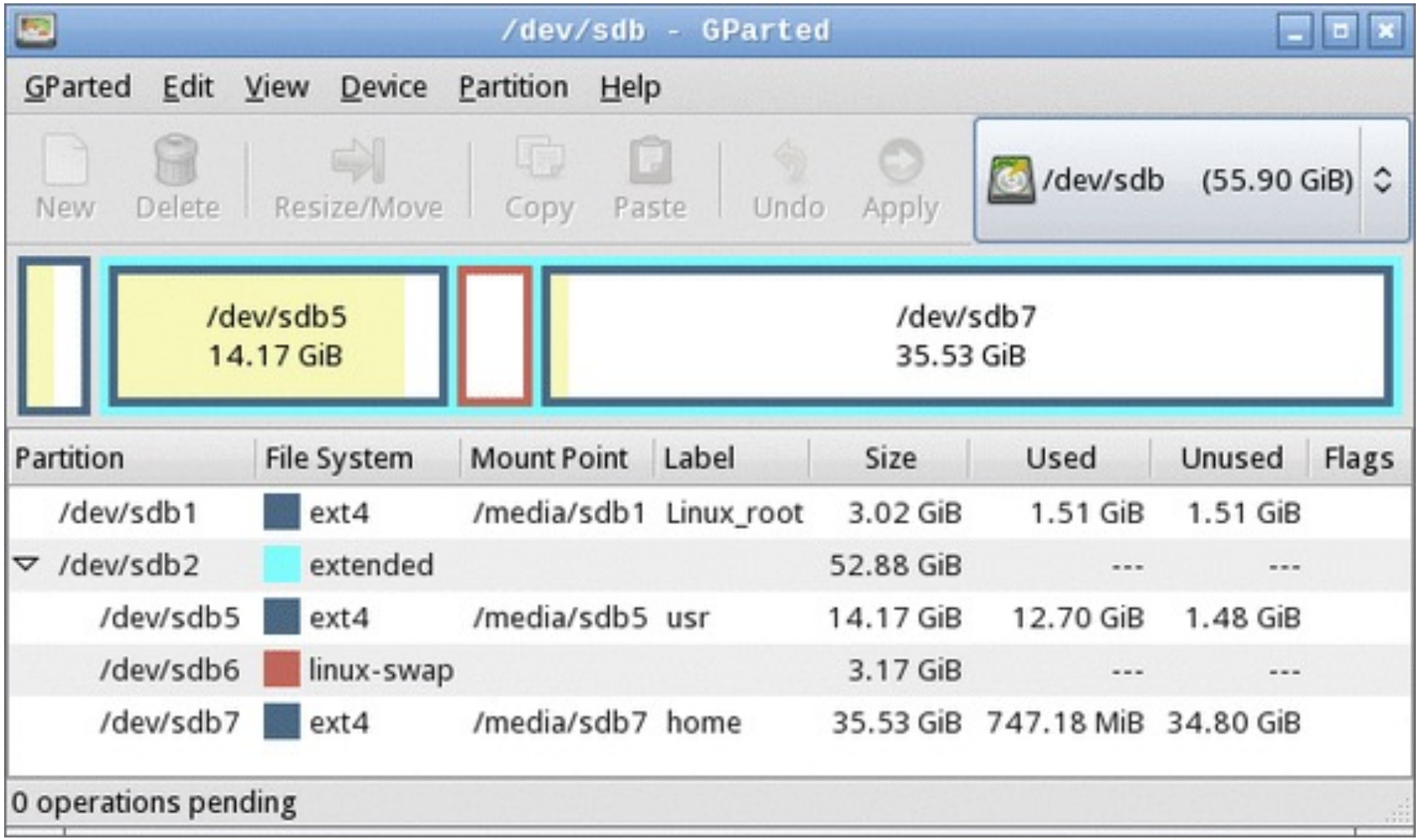


Example of file system (NTFS) in Windows

UNIX-based File System: ext2/ext3/ext4

- Extended file system (ext) for Unix/Linux.
- Versions: ext2/ext3/ext4
- Modern Linux uses ext4 as the default file system
- Advance features: journaling (with checksum support), support for large file size (16GB to 16TB)
- A directory may have a maximum of 64000 subdirectories

UNIX-based File System: ext2/ext3/ext4



Example of file system (ext4) in Linux

Plan

- File System Interface
- **File System Implementation**
  - *Overview*
  - ***File and Directory implementation***
  - *File System Structure*
- Disk Storage

Fundamentals

- Allocation methods deal with how to allocate disk blocks to files
  - ✓ Contiguous Allocation
  - ✓ Linked List Allocation
  - ✓ Indexed Allocation
  - ✓ Case study #1: Linked List Allocation with File-Table Allocation (Windows)
  - ✓ Case study #2: Multilevel Indexed Allocation (Unix-based)

Contiguous Allocation

- The entire file must be stored in contiguous blocks on the disk

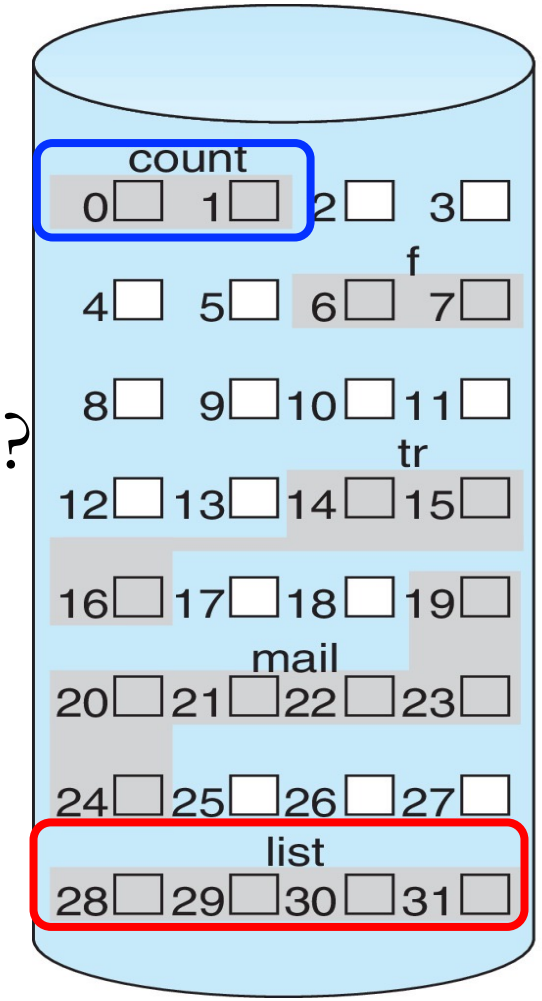
☺ Simple to implement

☺ High performance

☹ Fragmentation

☹ Find enough space for large files?

☹ Must know the final file size when creating a new file



directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Linked List Allocation

- The file can be stored in non-contiguous blocks on the disk, each of them contains the data and the pointer to the next block

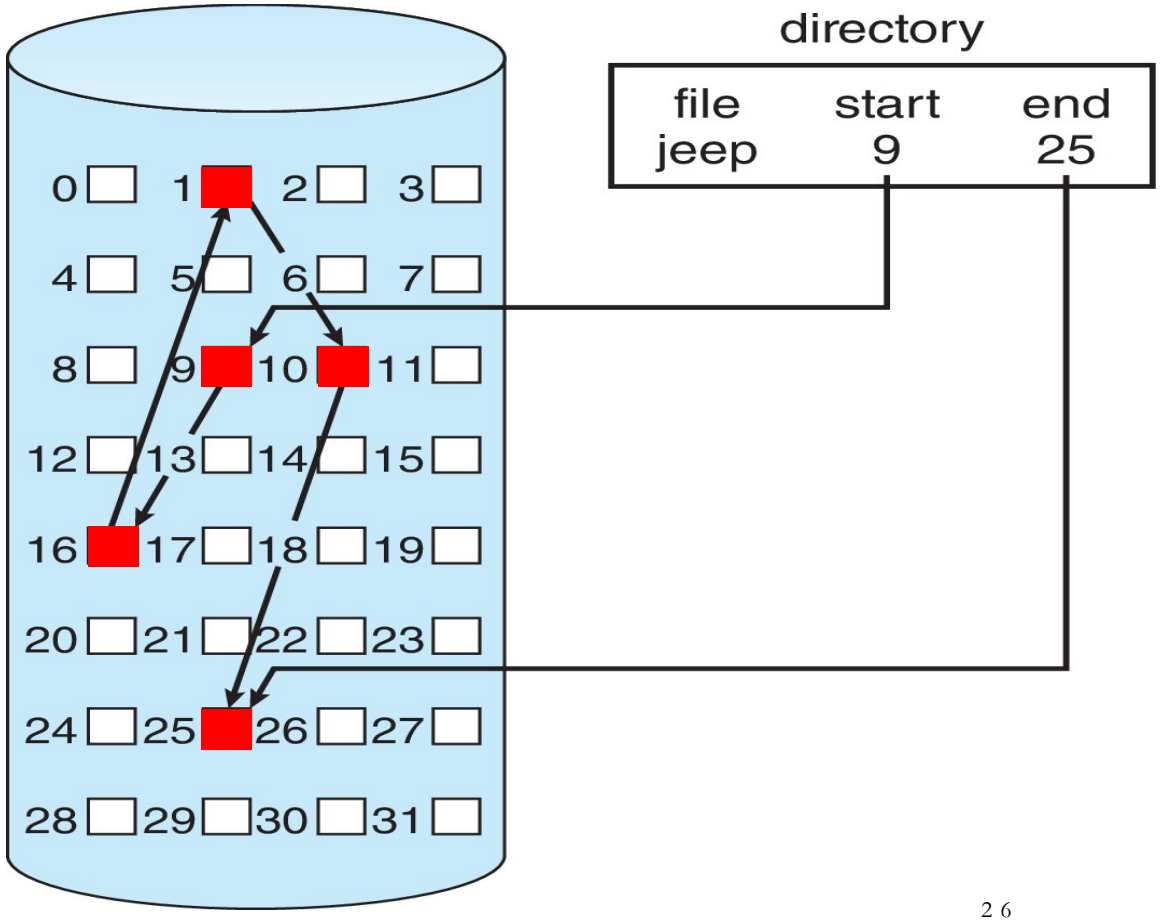
☺ Solving fragmentation problem

☺ Flexibility

☹ No direct random access

☹ Need space to store pointers

☹ What if a block is lost?



Indexed Allocation

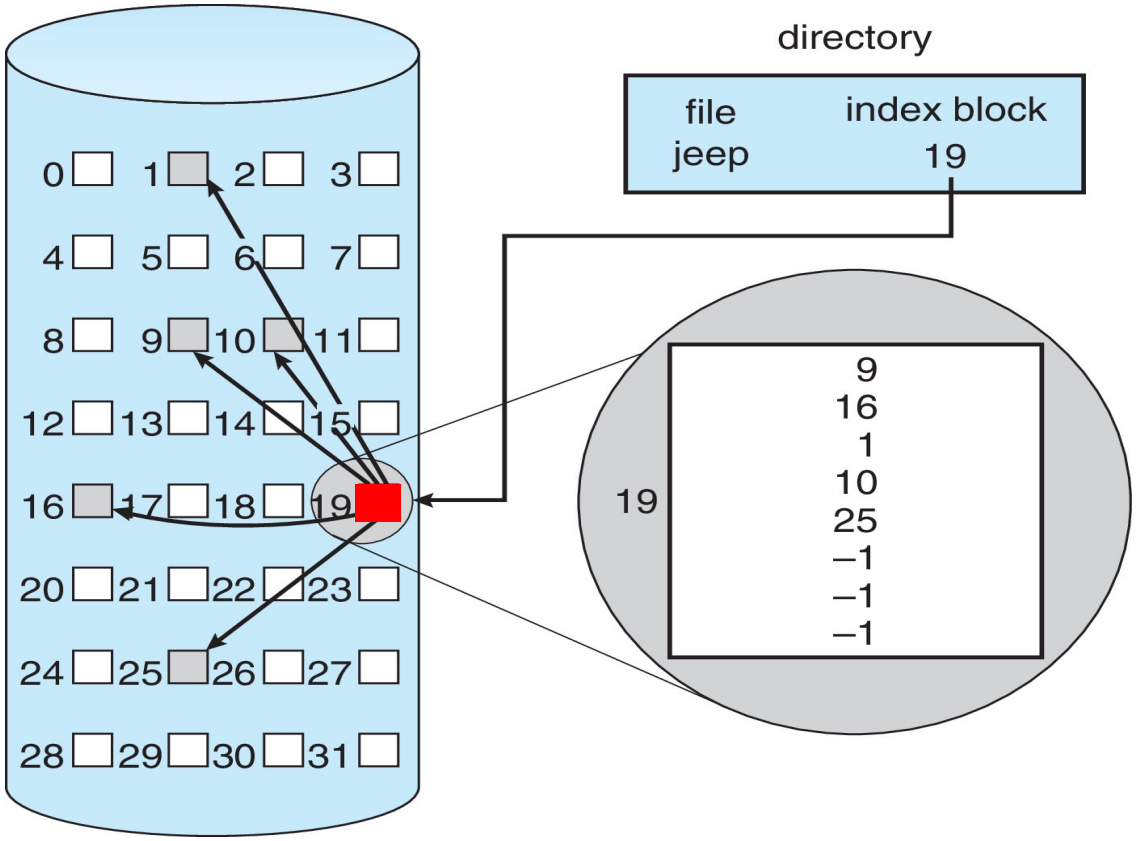
- The file can be stored in non-contiguous blocks on the disk
  - ✓ Data blocks for storing data
  - ✓ Index block for storing pointers to all data blocks

☺ Direct random access

☺ Solving lost block problem

☹ Need entire block for pointers

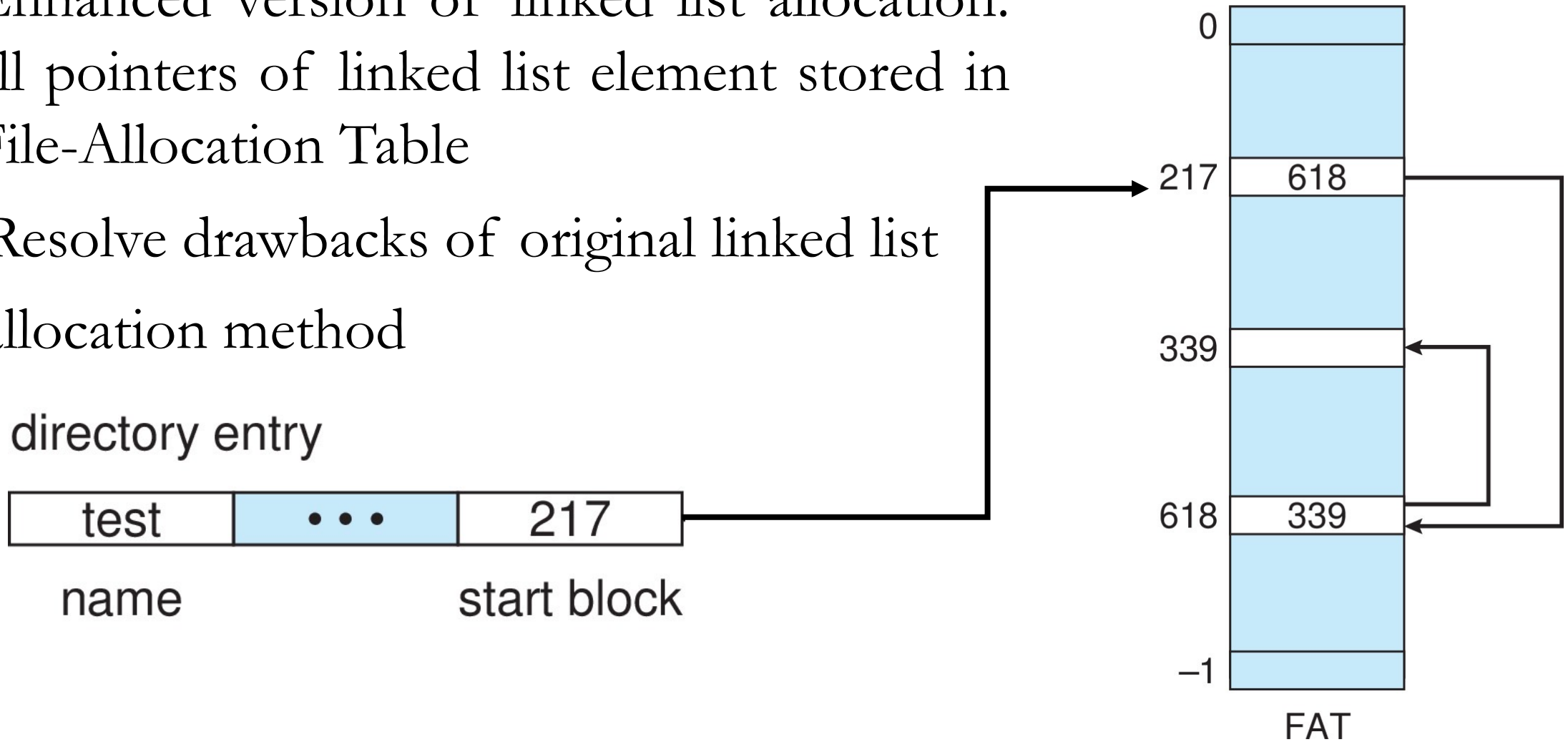
☹ Inefficiency for small files



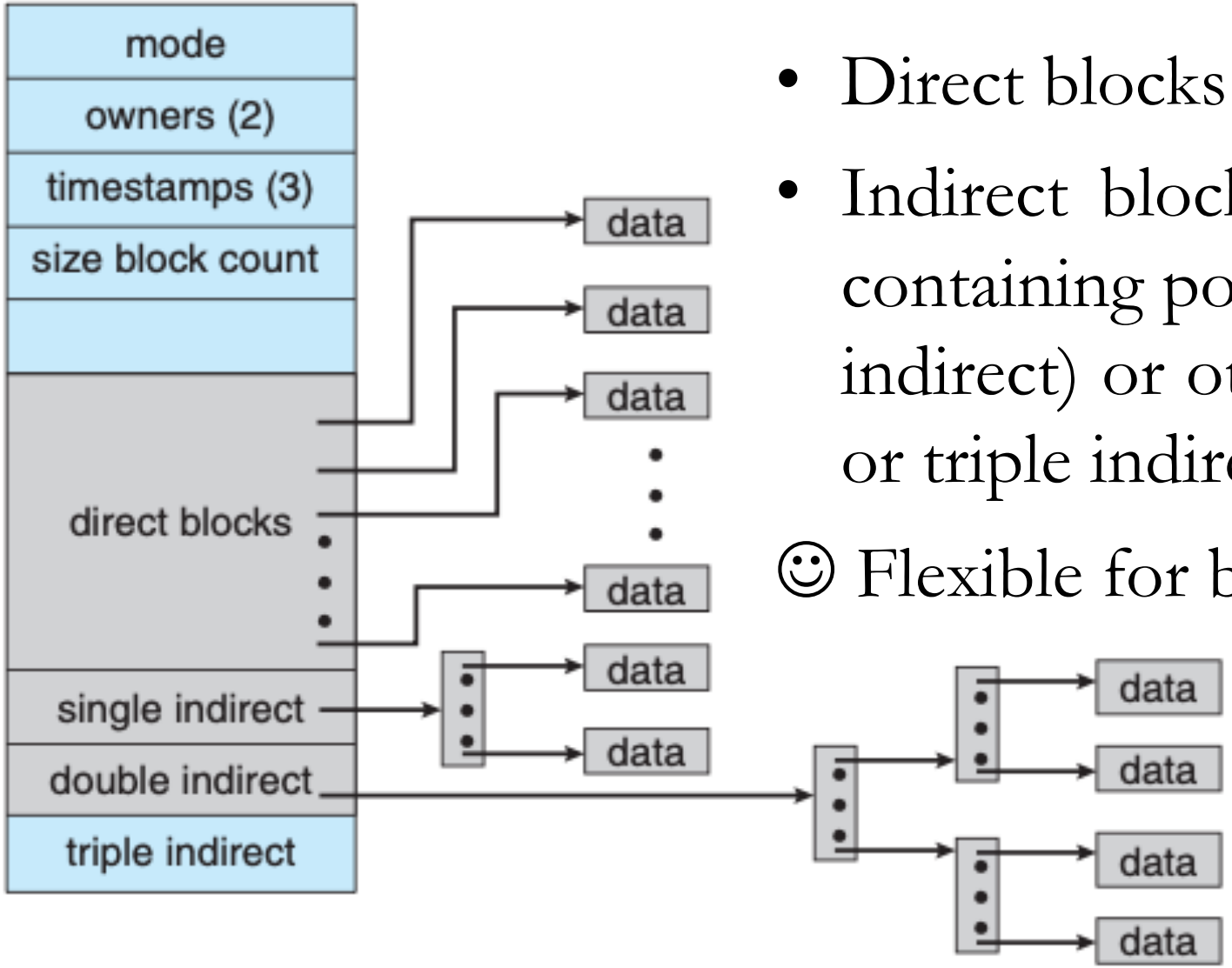
Windows: File-Allocation Table (FAT)

- Enhanced version of linked list allocation: all pointers of linked list element stored in File-Allocation Table

☺ Resolve drawbacks of original linked list allocation method



UNIX-based: Indexed Allocation (I-nodes)



- Multilevel indexed allocation
- Direct blocks point to data blocks
- Indirect blocks point to address blocks containing pointers to data blocks (single indirect) or other address blocks (double or triple indirect)

☺ Flexible for both large and small files

Free-space management

- **Bit map** or bit vector is a collection of bits, each of them corresponds to a disk block
  - ✓ 1: free-block; 0: allocated block
  - ✓ Example of a free-space bit map: 00011011100... (free-block: 3, 4, 6, 7, 8, ...)

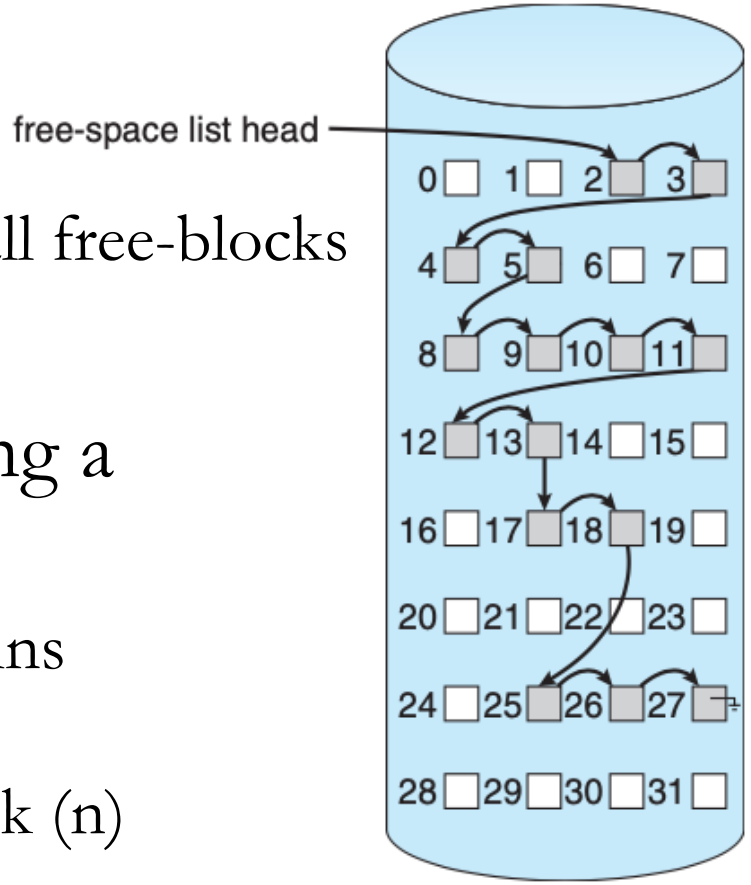
☺ Simplicity

☺ Efficiency

☹ Need to load the entire bit map into the main memory

Free-space management (cont.)

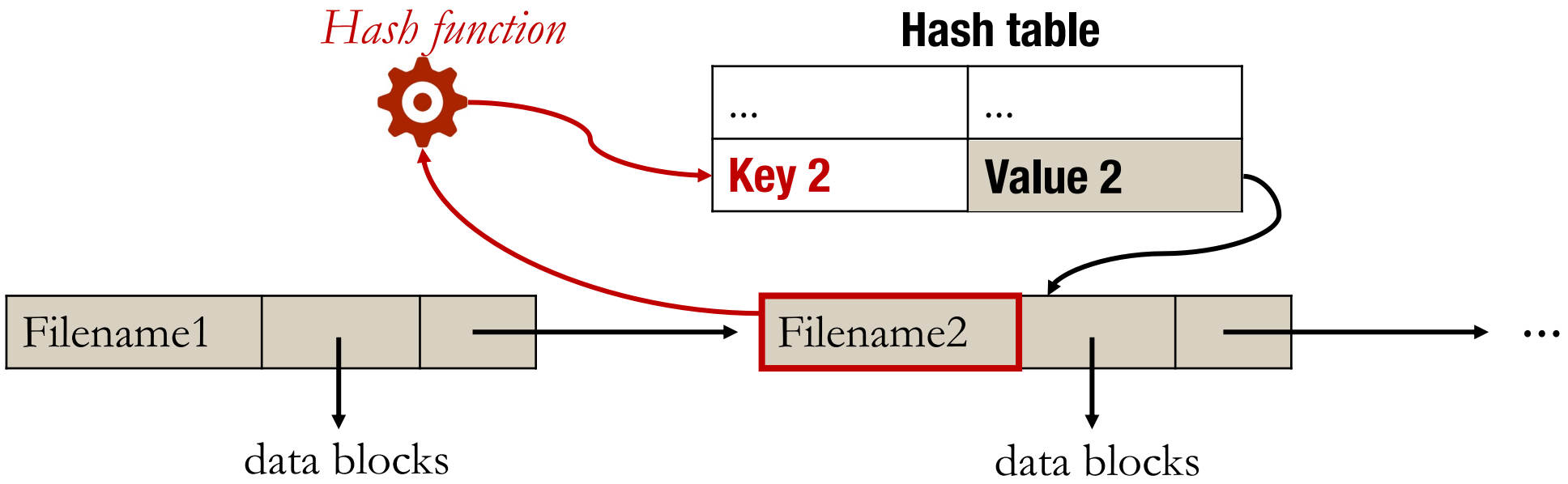
- **Linked list** of free-blocks: a free-block points to its next free-block
  - ☹ Free-block traversing
- **Grouping** n free-blocks together
  - ✓ First block of the group contains addresses of all free-blocks
  - ✓ The block (n-1) points to another n-free-blocks
- **Counting** n contiguous free-blocks following a block chosen as the first block.
  - ✓ Each element of the free-block linked list contains
    - its address
    - the number of its next contiguous free-block (n)



Example of linked list of free-blocks

Directory Implementation

- Linear linked list of filenames
  - Each entry in the list points to its data blocks, and the next entry
  - ☹ Linear searching for files
- Hashed table used along with a linked list to optimize searching
  - ✓ Each filename will be hashed and put in the hash table as a key
  - ✓ A value is associated with a key for finding the corresponding file



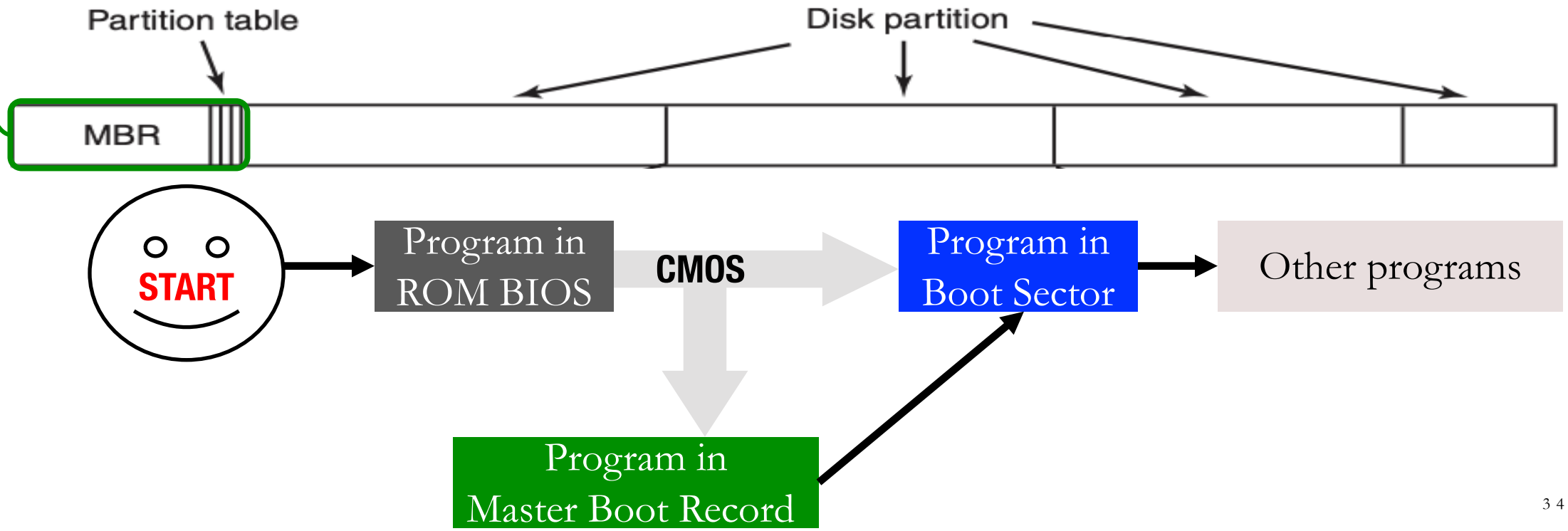
Plan

- File System Interface
- **File System Implementation**
  - *Overview*
  - *File and Directory implementation*
  - ***File System Structure***
- Disk Storage

Disk Layout

Master Boot Record

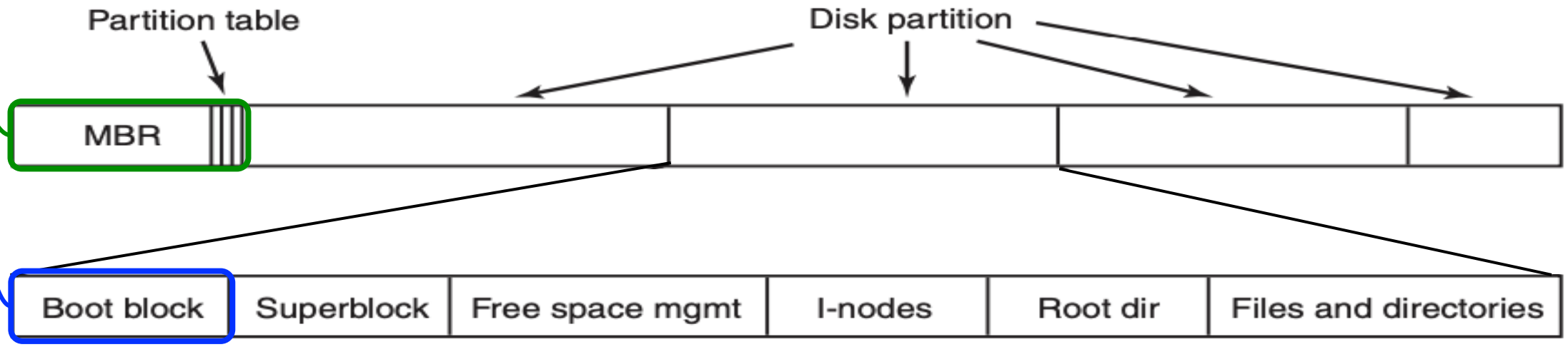
- Reside on the first sector of any hard disk
- Contain information for OS booting, also including the program that reads **boot sector** record of the disk partition containing OS



Disk Layout

Master Boot Record

- Reside on the first sector of any hard disk
- Contain information for OS booting, also including the program that reads **boot sector** record of the disk partition containing OS

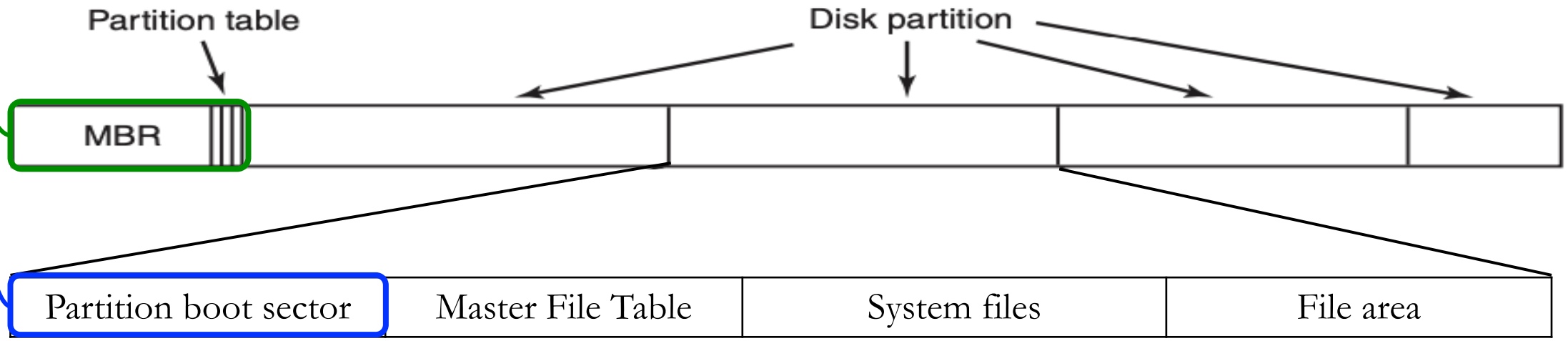


Example of disk layout (Unix)

Disk Layout

Master Boot Record

- Reside on the first sector of any hard disk
- Contain information for OS booting, also including the program that reads **boot sector** record of the disk partition containing OS



Example of disk layout (Windows)

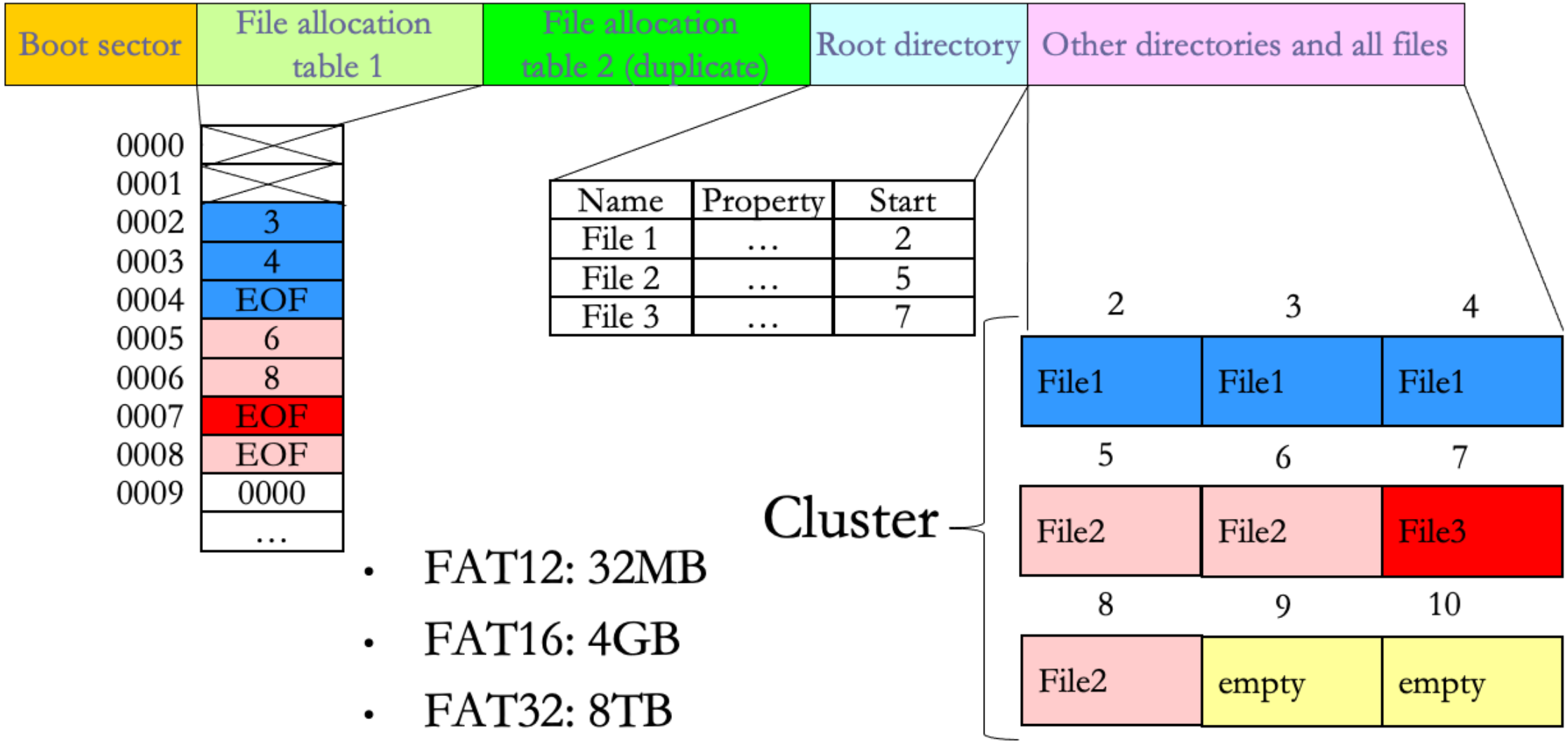
Master Boot Record

Address		Description		Size in bytes
Hex	Dec			
0000	0	Code Area		max. **446**
01B8	440	Optional Disk signature		4
01BC	444	Usually Nulls; 0x0000		2
01BE	446	**Table of primary partitions** (Four 16-byte entries, IBM Partition Table scheme)		**64**
01FE	510	55h	MBR signature; 0xAA55[1]	**2**
01FF	511	AAh		
MBR, total size: 446 + 64 + 2 =				**512**

Master Boot Record (cont.)

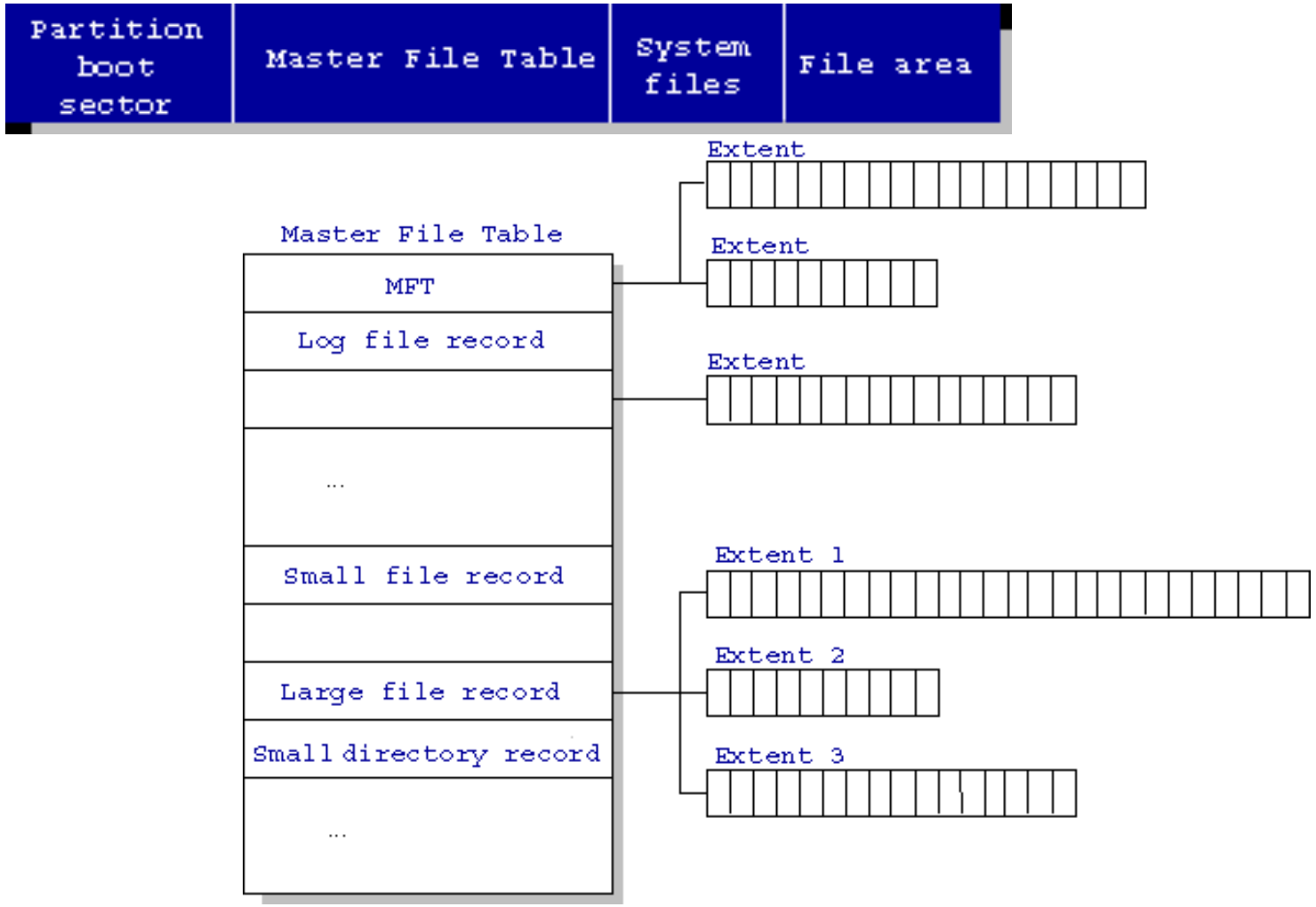
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000000000	90	E9	7D	01	FA	33	C0	8E	D0	8E	C0	8E	D8	BC	00	7C	▌é}.ú3À▌Ð▌À▌Ø¼.|
0000000010	8B	F4	FB	BF	00	06	B9	00	01	F3	A5	BB	20	06	FF	E3	▌ôû¿..¹..ó¥» .ÿã
0000000020	90	90	BE	7D	07	81	3C	AA	55	75	11	E8	58	00	73	0C	▌▌¾}.▌<ªUu.èX.s.
0000000030	E8	65	00	72	07	E8	B1	00	72	3B	EB	2C	BE	7D	07	C7	èe.r.è±.r;ë,¾}.Ç
0000000040	04	00	00	BA	80	00	BE	BE	07	B9	04	00	F6	04	80	75	...º▌.¾¾.¹..ö.▌u
0000000050	07	83	C6	10	E2	F6	EB	1D	8A	74	01	8B	4C	02	BB	00	.▌Æ.âöë.▌t.▌L.».
0000000060	7C	B8	01	02	CD	13	72	0D	81	3E	FE	7D	55	AA	75	05	|¸..Í.r.▌>þ}Uªu.
0000000070	EA	00	7C	00	00	BE	6A	07	AC	0A	C0	74	FE	BB	07	00	ê.|..¾j.¬.Àtþ»..
0000000080	B4	0E	CD	10	EB	F2	BB	00	7E	C6	07	13	C6	47	01	00	´.Í.ëò».~Æ..ÆG..
0000000090	B2	80	B8	00	E0	CD	13	C3	BF	00	7E	BA	F0	01	B3	A0	²▌¸.àÍ.Ã¿.~º ð.³
00000000A0	E8	84	00	72	0C	B1	01	E8	48	00	72	05	E8	19	00	73	è▌.r.±.èH.r.è..s
00000000B0	16	F6	C3	10	75	05	80	CB	10	EB	E5	81	FA	70	01	74	.öÃ.u.▌Ë.ëå▌úp.t
00000000C0	05	BA	70	01	EB	D8	F9	C3	81	BD	FE	01	55	AA	75	17	.ºp.ëØùÃ▌½þ.Uªu.
00000000D0	8B	75	02	81	FE	BE	01	77	0E	03	F7	81	3C	AA	55	75	▌u.▌þ¾.w..÷▌<ªUu
00000000E0	06	F6	44	02	01	75	01	F9	C3	BF	00	7C	B1	0A	E8	01	.öD..u.ùÃ¿.|±.è.
00000000F0	00	C3	52	57	83	C2	02	B0	01	EE	42	8A	C1	EE	42	32	.ÃRW▌Å.°.îB▌ÁîB2
0000000100	C0	EE	42	EE	42	8A	C3	EE	42	B0	20	EE	E8	33	00	EC	ÀîBîB▌ÃîB° îè3.ì
0000000110	24	FD	3C	58	75	0D	83	EA	07	B9	00	01	FA	F3	6D	FB	$ý<Xu.▌ê.¹..úómû
0000000120	F8	EB	01	F9	5F	5A	C3	52	83	C2	07	EC	A8	80	75	0F	øë.ù_ZÃR▌Å.ì¨▌u.
0000000130	4A	8A	C3	EE	42	EC	24	D0	3C	50	75	03	F8	EB	01	F9	J▌ÃîBì$Ð<Pu.øë.ù
0000000140	5A	C3	51	8B	0E	6C	04	83	C1	12	81	C2	FF	01	EC	8A	ZÃQ▌.l.▌Á.▌Åÿ.ì▌
0000000150	E0	80	E4	D8	80	FC	58	74	06	3B	0E	6C	04	75	EF	81	à▌äØ▌üXt.;.l.uï▌
0000000160	EA	FF	01	B9	00	20	E2	FE	59	C3	0D	0A	45	72	72	6F	êÿ.¹. âþYÃ..Erro
0000000170	72	20	4C	6F	61	64	69	6E	67	20	4F	53	00	55	AA	00	r Loading OS.Uª.
0000000180	00	E9	80	FE	00	00	00	00	68	00	00	00	00	00	00	00	.é▌þ....h.....
0000000190	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000001A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000001B0	00	00	00	00	00	00	00	00	00	0A	01	0A	00	00	80	01	.....▌.
00000001C0	01	00	0C	FE	FF	FF	3F	00	00	00	C1	52	A8	04	00	00	...þÿÿ?...ÁR¨...
00000001D0	C1	FF	0F	FE	FF	FF	00	53	A8	04	C1	91	A8	04	00	00	Áÿ.þÿÿ.S¨.Á´¨...
00000001E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	.....
00000001F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	55	AA	.....Uª

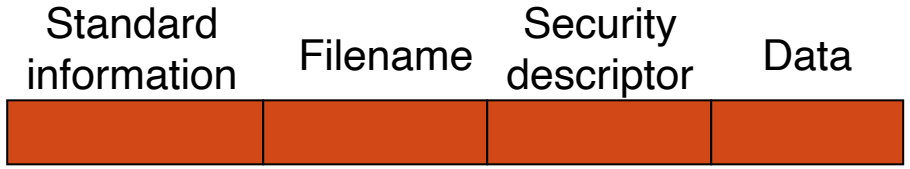
FAT Structure



- FAT12: 32MB
- FAT16: 4GB
- FAT32: 8TB

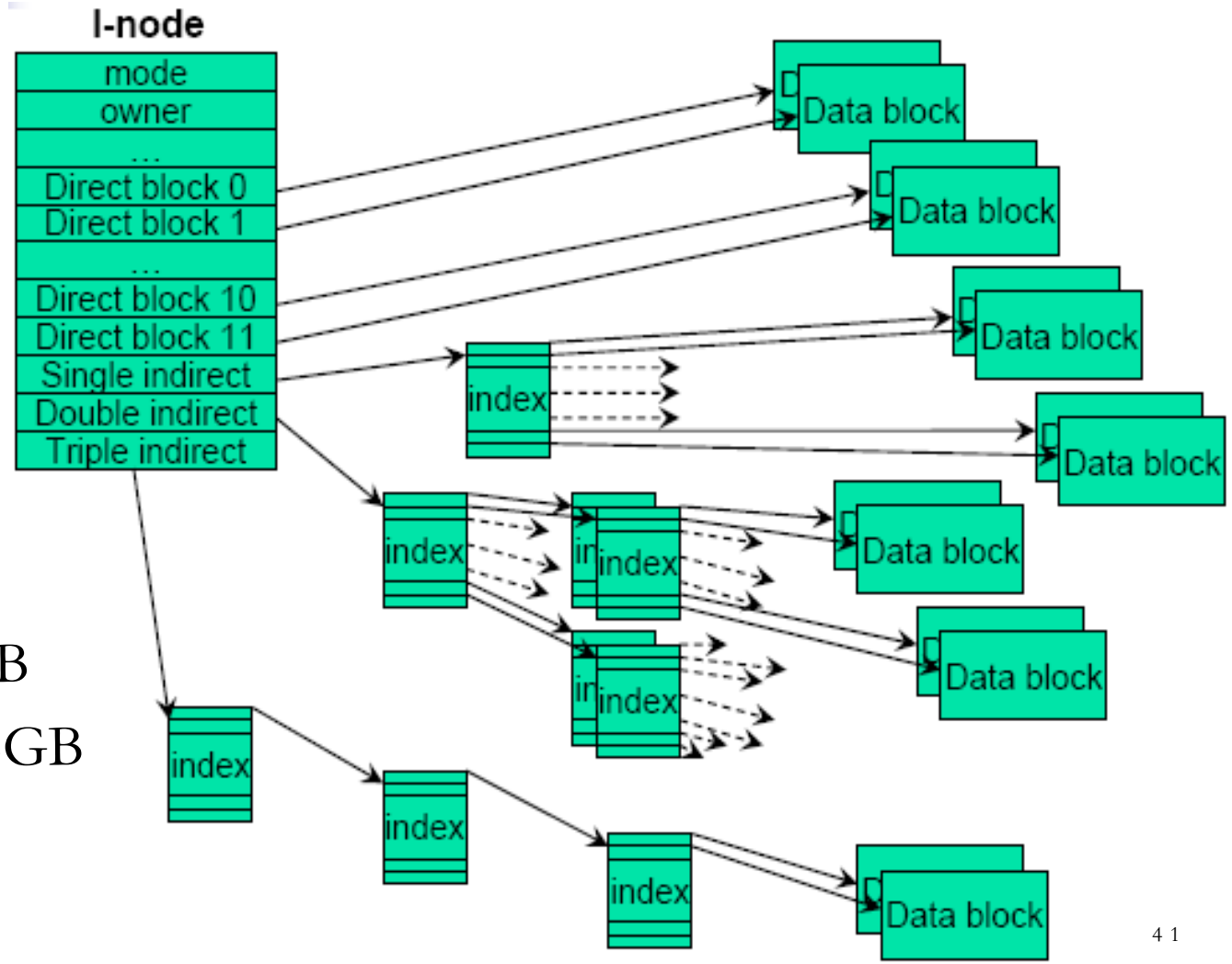
NTFS Structure





UNIX I-node Structure

boot block	super block	I-nodes	files and directories



- Single indirect: 256KB
- Double indirect: 256*256 = 65 MB
- Triple indirect: 256*256*256 = 16GB

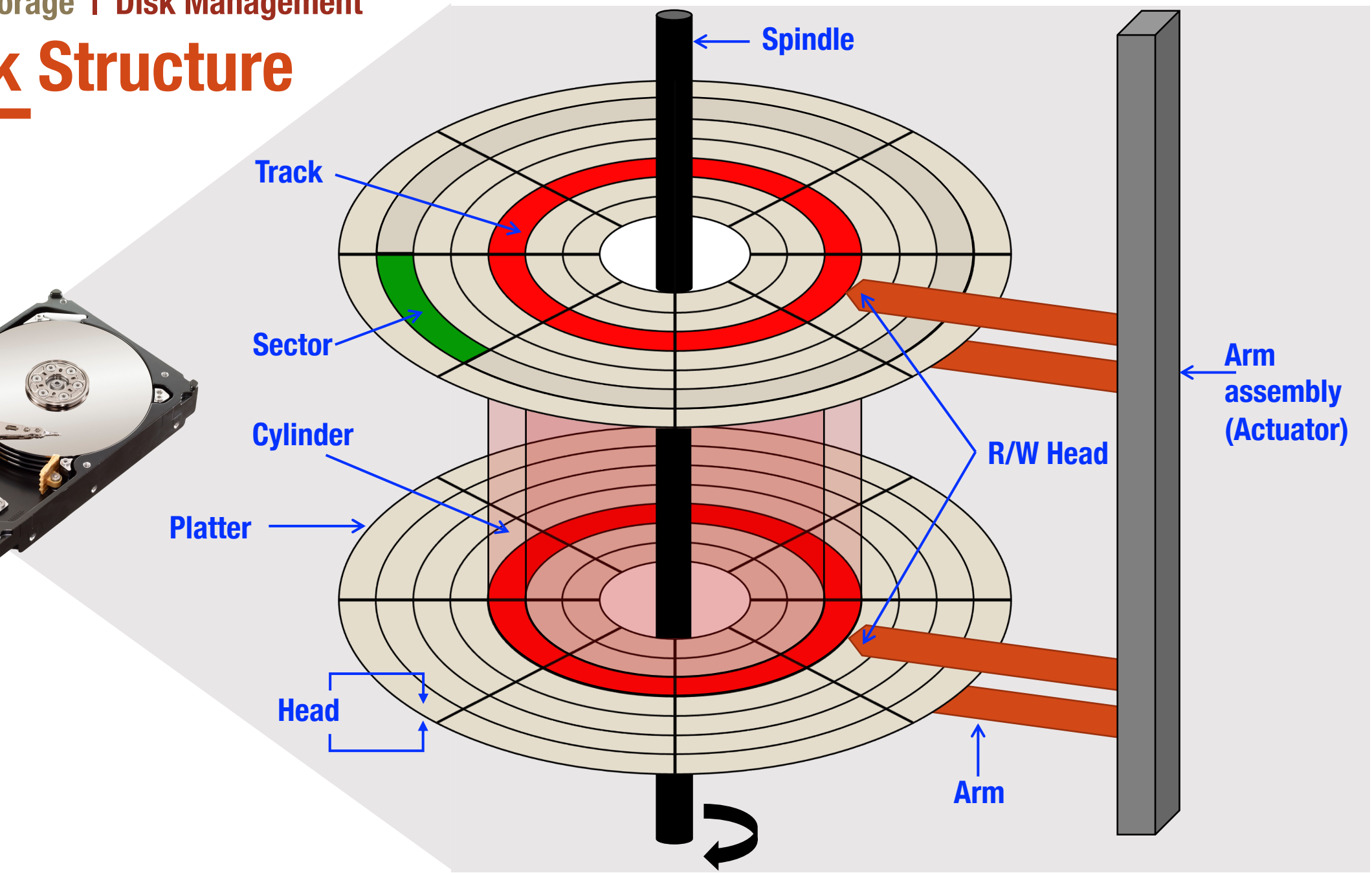
Plan

- File System Interface
- File System Implementation
- **Disk Storage**
  - *Disk Management*
  - *Disk Scheduling Algorithms*

Plan

- File System Interface
- File System Implementation
- **Disk Storage**
  - ***Disk Management***
  - *Disk Scheduling Algorithms*

Disk Structure



Disk Structure

8 sectors (of 512 bytes) per track

4 tracks (or heads) per cylinder

7 cylinders (⇔ 7 tracks per head)

➔ Total no. of sector: 224

➔ Disk capacity: ~ 112KB

Disk Formatting

Performed by hard disk manufactures

Performed by users via specific software or an operating system

Low-level formatting

Creation of sectors

- Divide tracks into sectors with a special data structure

Partition

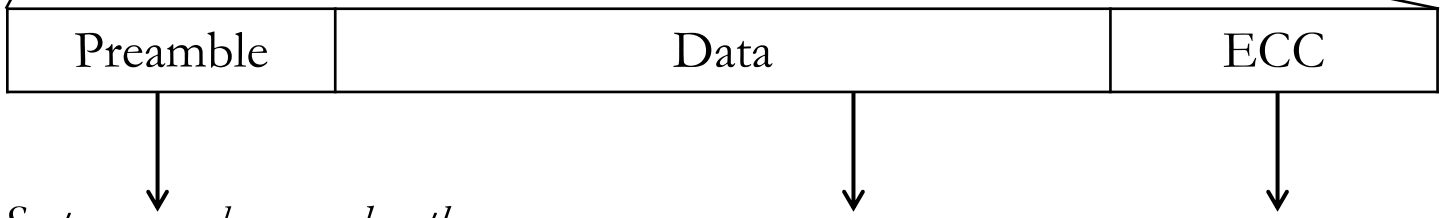
Creation of partitions

- Group cylinders into one or more partitions
- Check for track or sector errors

High-level formatting

Creation of file systems

- Create file systems on partitions while installing an OS

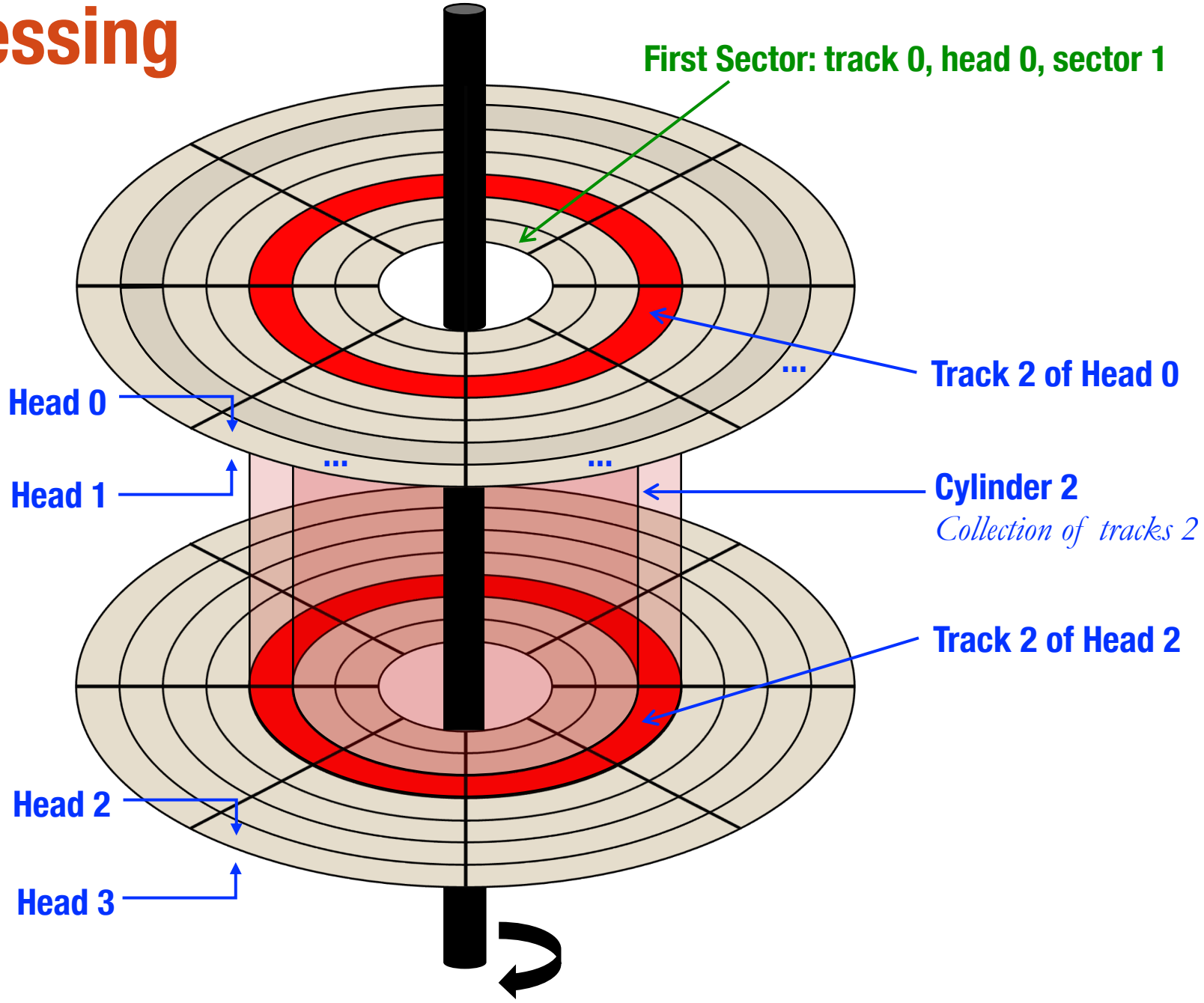


Sector number and other information for recognizing the start of the sector

Most hard disks use 512-byte sectors

Error-correcting code for recognizing read errors and for recovering the sector from errors

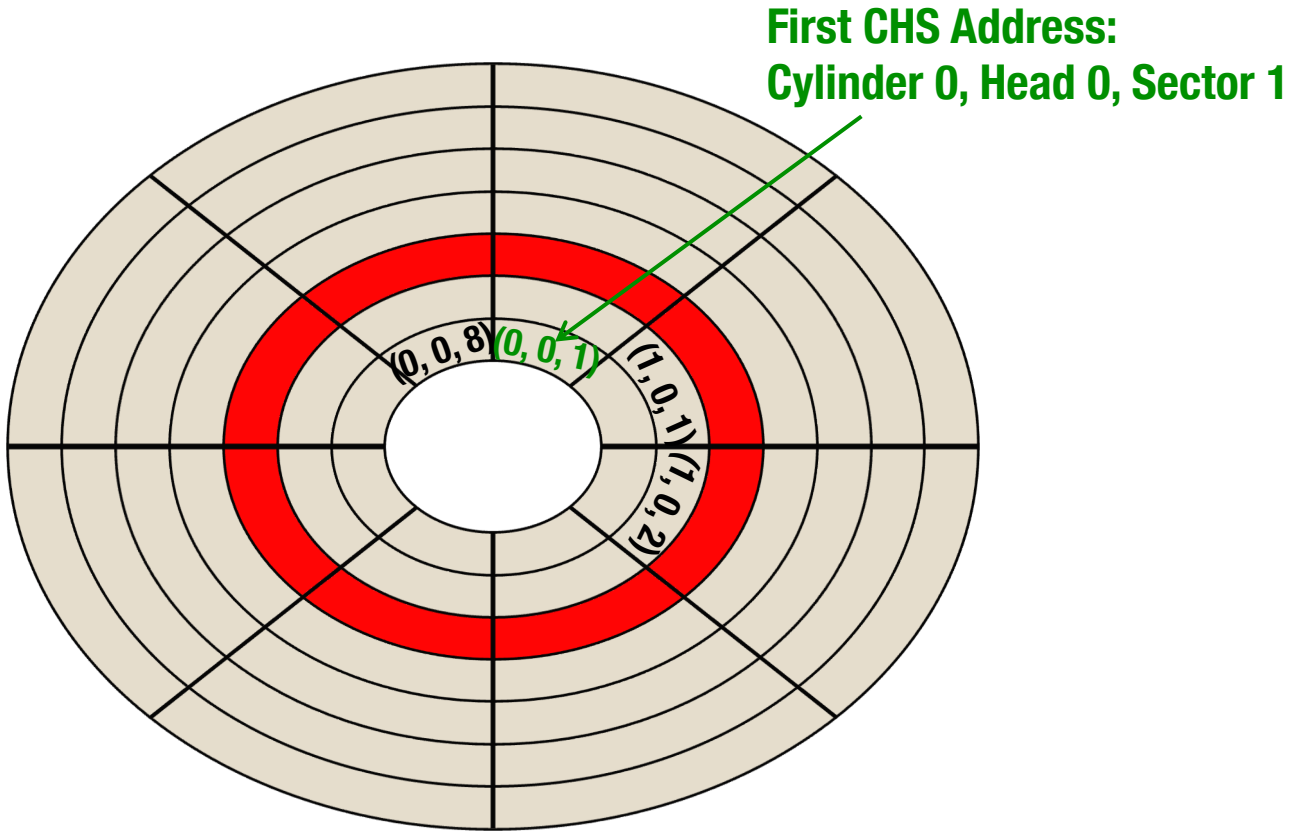
Disk Accessing



Disk Accessing

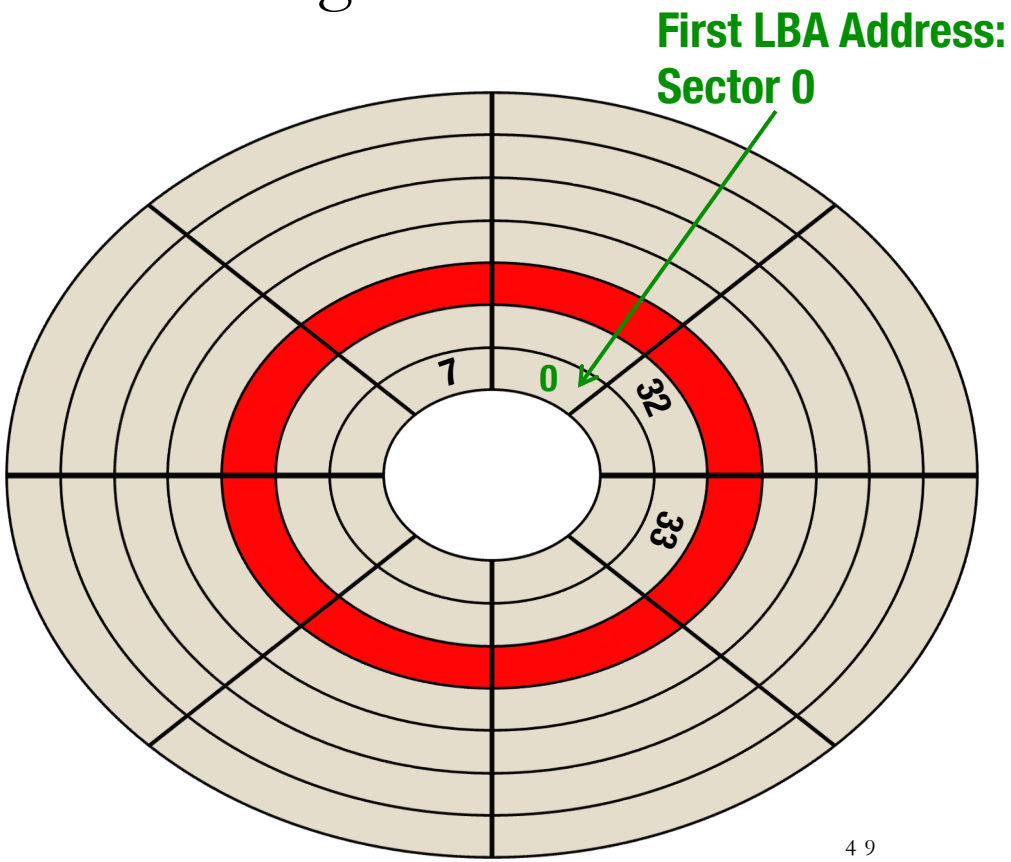
CHS (Cylinder-Head-Sector)

Disk blocks located by a tuple of (cylinder|track, head, sector)



LBA (Logical Block Addressing)

Disk blocks located by an integer index starting at 0



CHS – LBA Mapping

CHS to LBA address

$$LBA = (C * N_{heads} + H) * N_{sectors} + (S - 1)$$

LBA to CHS address

$$C = LBA / (N_{heads} * N_{sectors})$$
$$H = (LBA / N_{sectors}) \% N_{heads}$$
$$S = (LBA \% N_{sectors}) + 1$$

Where:

- C: cylinder | track number
- H: head number
- S: sector number
- N_{heads}: no. of heads | tracks per cylinder
- $N_{sectors}$: no. of sectors per track

Assume a hard disk having:

- 4 heads/cylinder (N_{heads})
- 8 sectors/track ($N_{sectors}$)

Example of LBA – CHS Mapping

LBA	C	H	S
0	0	0	1
1	0	0	2
2	0	0	3
...	...	...	...
7	0	0	8
...			
32	1	0	1
...	...	...	...
223	6	3	8

Plan

- File System Interface
- File System Implementation
- **Disk Storage**
  - *Disk Management*
  - ***Disk Scheduling Algorithms***

Disk Access Time

Most dominant

Disk Access Time = Seek Time + Rotational Time + Data Transfer Time

Time to move Read/Write Head to the desired track/cylinder

Time to rotate the desired sector to the Read/Write Head

Time to transfer data from the disk = Transferred Data/Transfer Rate

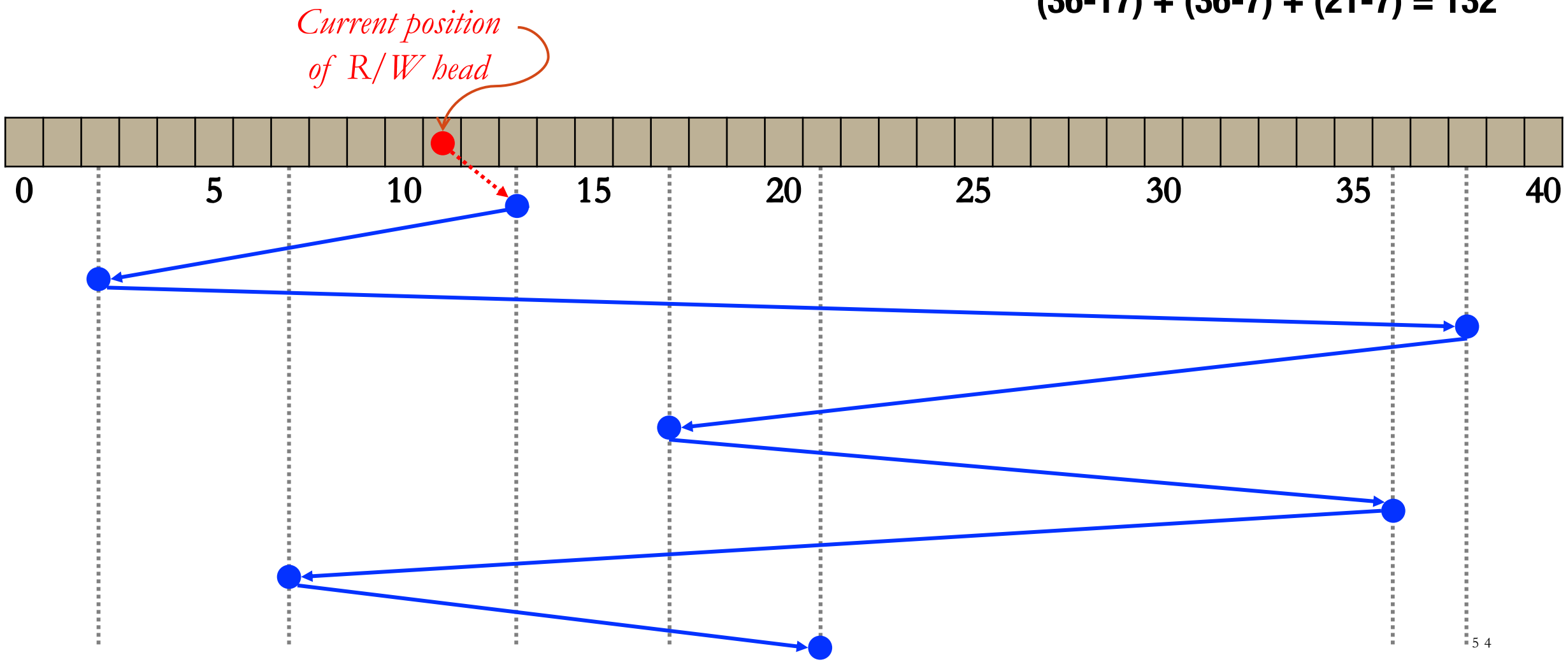
Disk Scheduling

- Disk Scheduling Algorithms
  - ✓ FCFS (First-Come First-Served)
  - ✓ SSTF (Shortest Seek Time First)
  - ✓ SCAN (Elevator Algorithm)
  - ✓ C-SCAN (Circular SCAN)
  - ✓ LOOK (an optimized version of SCAN)
  - ✓ C-LOOK (Circular LOOK)

FCFS (First-Come First-Served)

Disk requests: 13, 2, 38, 17, 36, 7, 21

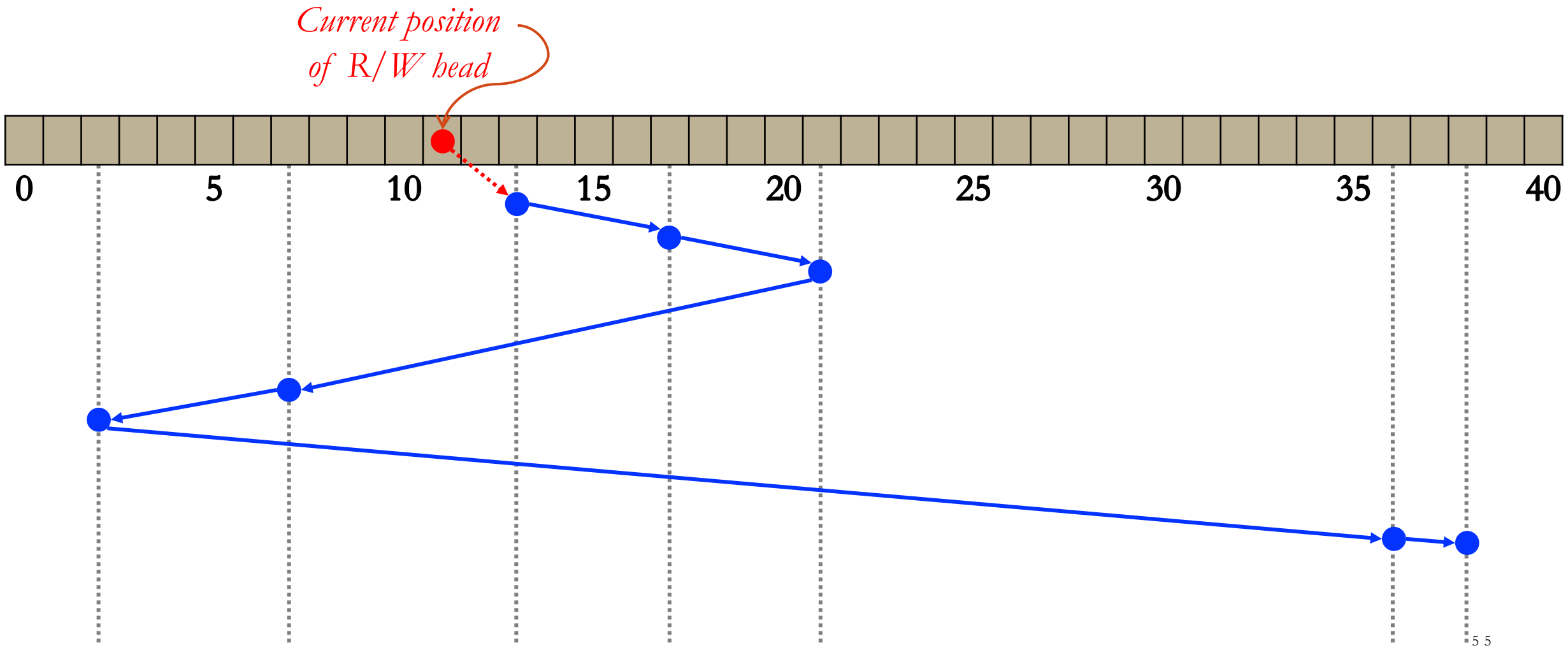
No. track movement = (13-11) + (13-2) + (38-2) + (38-17) + (36-17) + (36-7) + (21-7) = 132



SSTF (Shortest Seek Time First)

Disk requests: 13, 2, 38, 17, 36, 7, 21

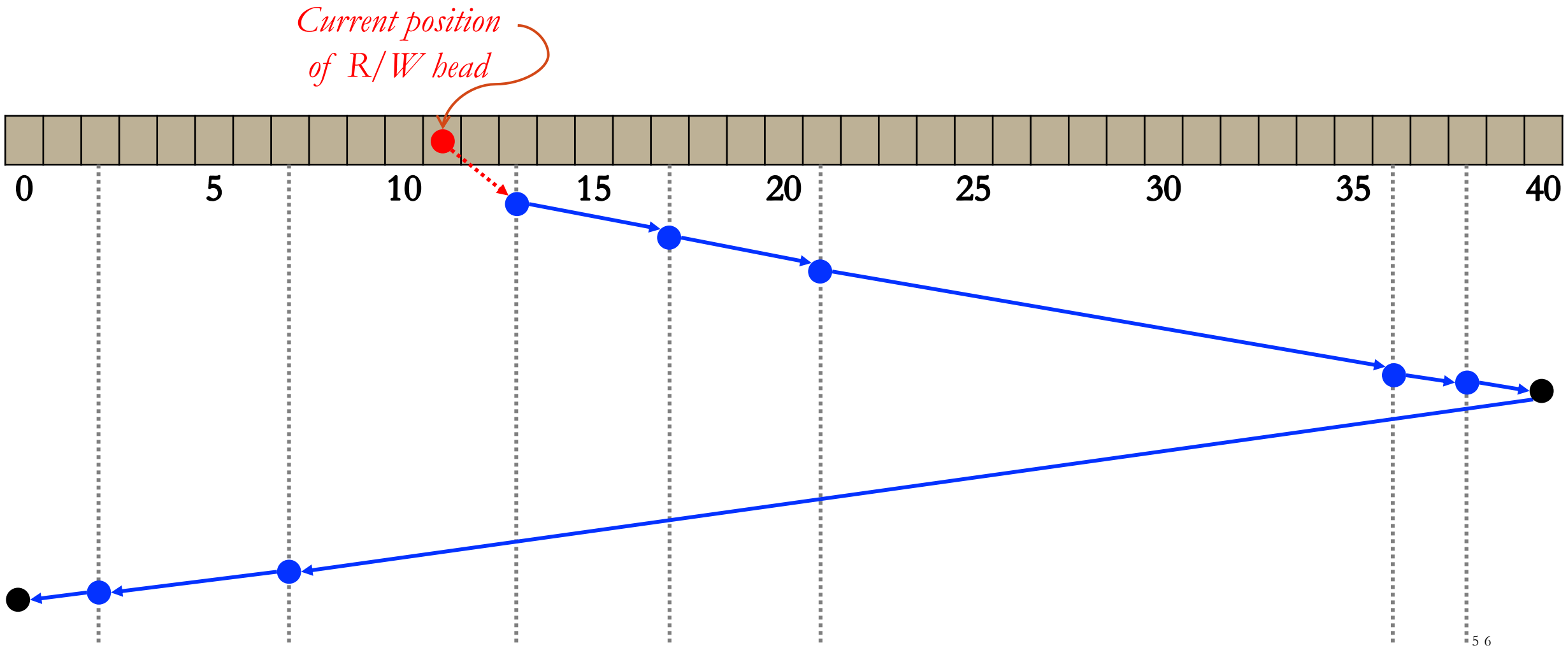
No. track movement = (21-11) + (21-2) + (38-2) = 65



SCAN

Disk requests: 13, 2, 38, 17, 36, 7, 21

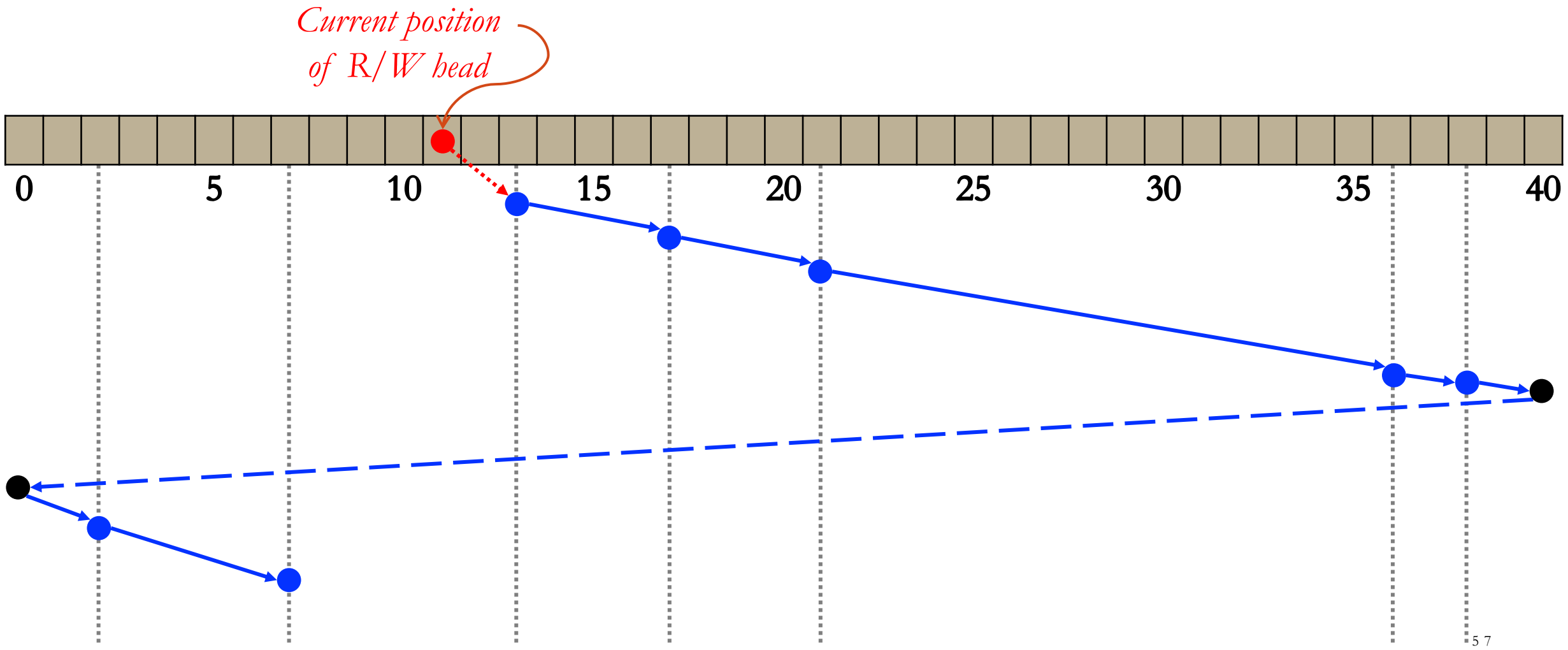
No. track movement = (40-11) + (40-2) = 67



C-SCAN

Disk requests: 13, 2, 38, 17, 36, 7, 21

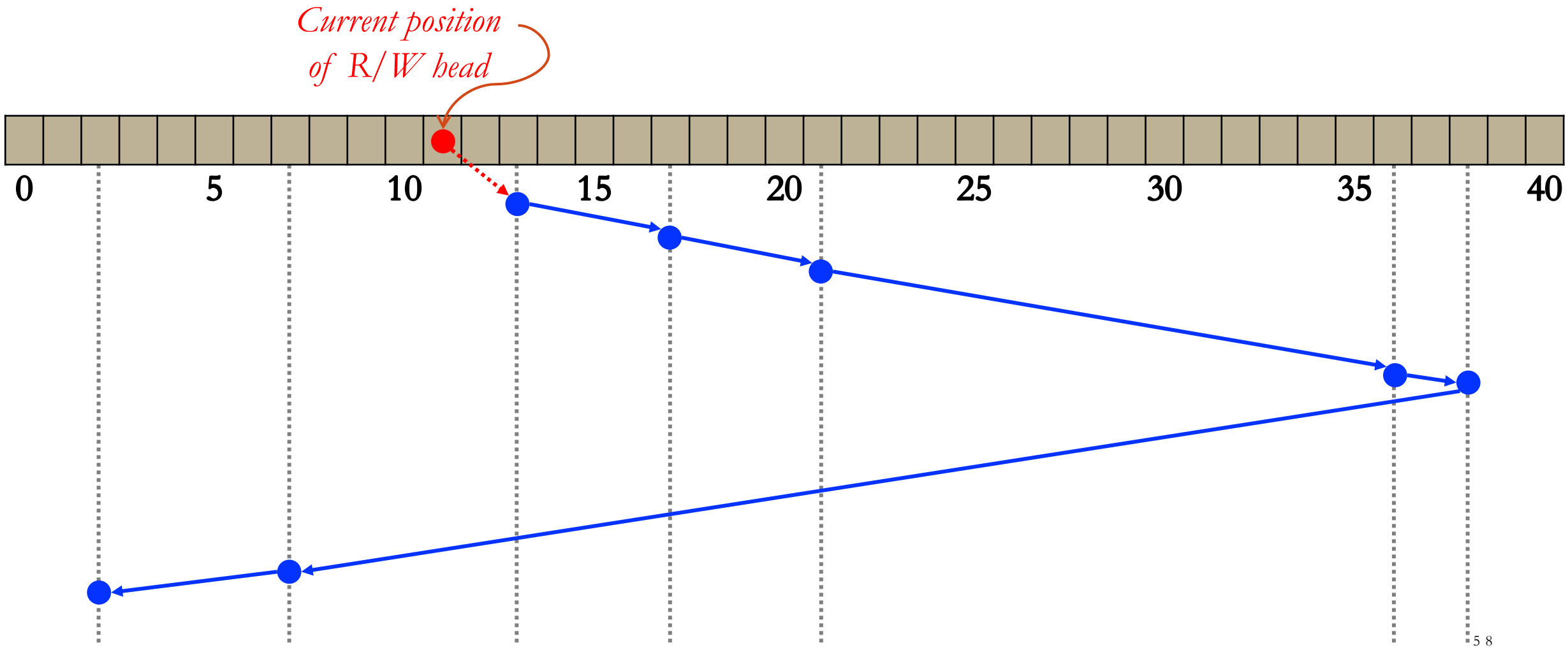
No. track movement = (40-11) + (40-0) + (7-0) = 76



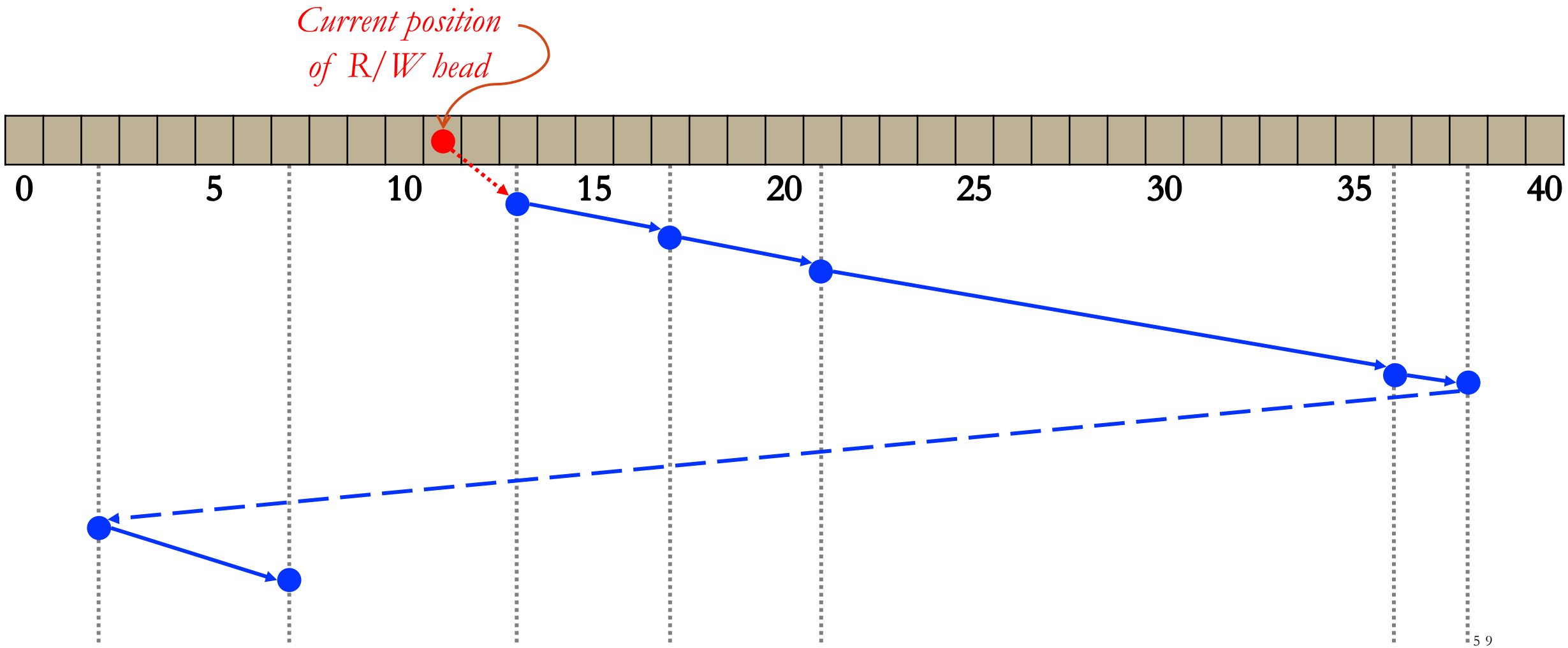
LOOK

Disk requests: 13, 2, 38, 17, 36, 7, 21

No. track movement = (38-11) + (38-2) = 63



C-LOOK

Disk requests: 13, 2, 38, 17, 36, 7, 21

No. track movement = (38-11) + (38-2) + (7-2) = 68