File systems: case studies

Hung-Wei Tseng

Recap: How your application reaches storage

User

Applications

data   fread/fwrite — input.bin/output.bin

I/O libraries   **Buffer**

fread/fwrite — input.bin/output.bin

File system

read/write — 0, 512, 4096, … (block address)

Buffer   **Device independent I/O interface (e.g. ioctl)**

Kernel data

read/write — block addresses

Device Driver   **Device Driver**   **Device Driver**

read/write — block addresses

Device Controller   **Device Controller**   **Device Controller**

Hardware

Device #1   **Device #2**   **Device #3**   **Device #4**
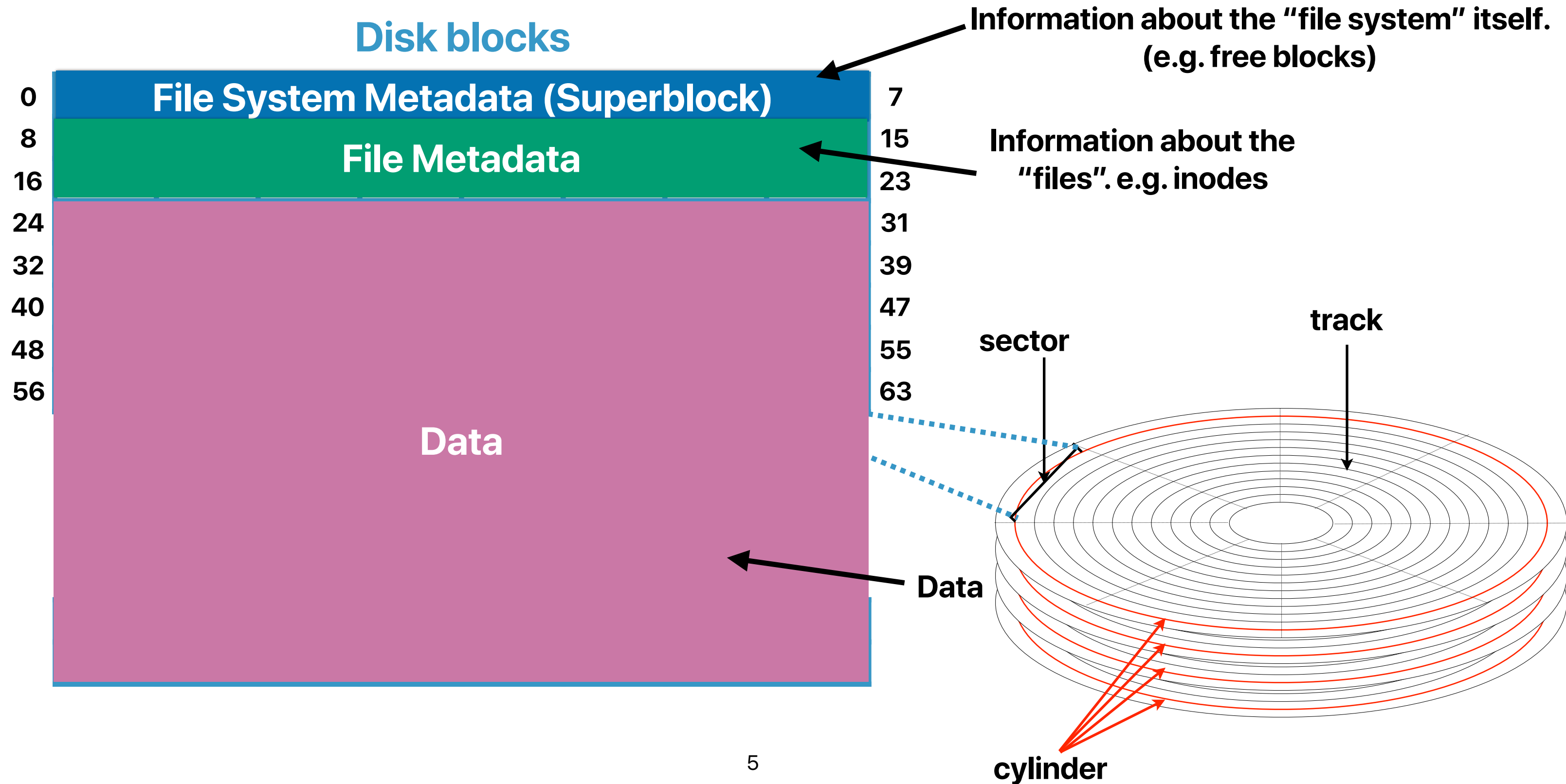
Recap: The "file" abstraction

- What is a file?
  - A logical unit of storage (e.g. an mp3), a device or a directory
  - Operations: open, close, read, write
- What is a file system?
  - A logically-structured collection of files
  - Defines the namespace of a file
  - Provides persistence, access control, and other protection/security mechanisms
- Files / File System provides an abstraction for secondary storage

Abstractions in operating systems

- Process — the abstraction of a von Neumann machine

- Thread — the abstraction of a processor

- Virtual memory — the abstraction of memory

- File system — the abstraction of space/location on a storage device, the storage device itself, as well as other peripherals

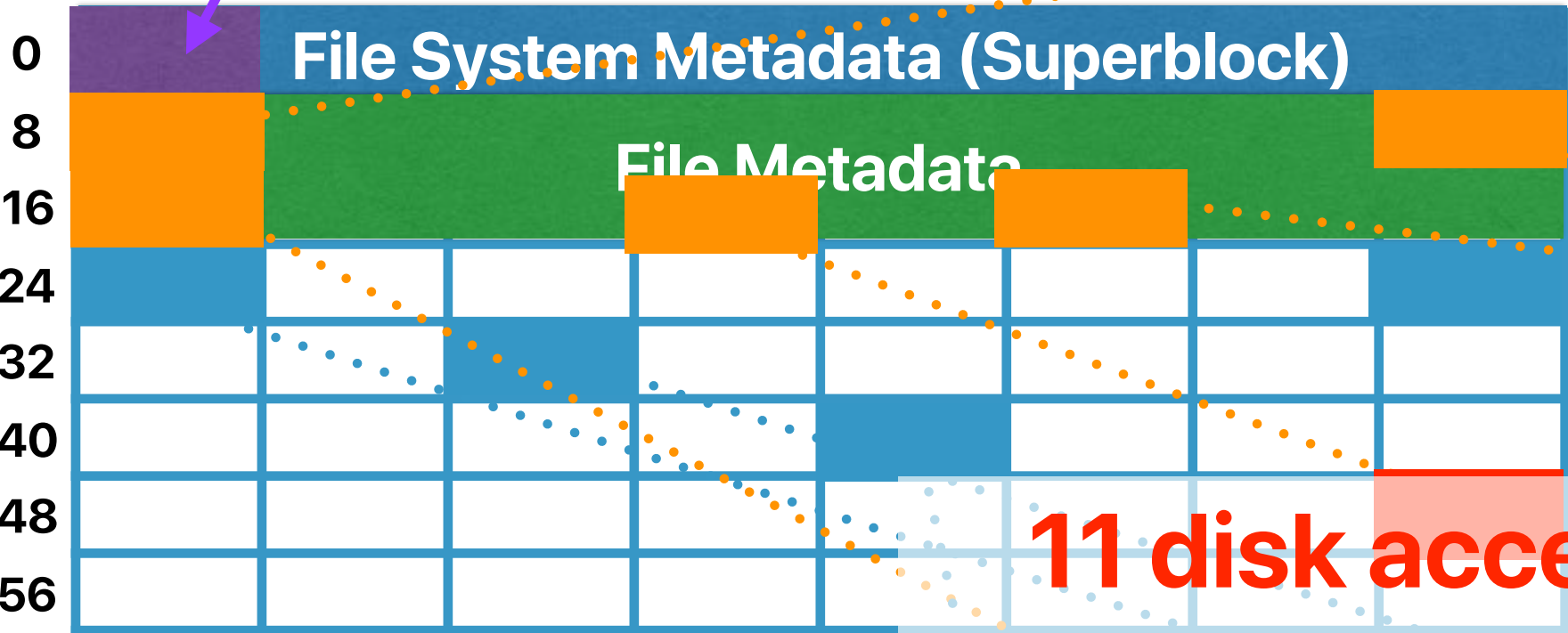# Recap: How the original UNIX file system use disk blocks

Disk blocks

**Information about the "file system" itself.
(e.g. free blocks)**

0	**File System Metadata (Superblock)**	7
8	**File Metadata**	15
16		23

**Information about the
"files". e.g. inodes**

24		31
32		39
40		47
48		55
56		63

Data

sector

track

Data

cylinder

How to reach /home/hungwei/CS202/foo.c

Superblock

Disk blocks

index node (inode)

	File System Metadata (Superblock)		
0			7
8	File Metadata		15
16			23
24			31
32			39
40			47
48			55
56			63

11 disk accesses — each takes more than 10ms — in getting a file!

inode 1	
owner_id	0
permission	755
type	dir
address	24
...	

inode 15	
owner_id	0
permission	755
type	dir
address	31
...	

inode 21	
owner_id	0
permission	755
type	dir
address	34
...	

```
#include
<stdio.h>
.
.
.
.
```

home	
tyler	20
hungwei	21

CS202	
bar.c	18
foo.c	19

inode 16	
owner_id	0
permission	755
type	dir
address	44
...	

/	
usr	13
var	14
home	15

hungwei	
CS202	16
Dropbox	17

inode 19	
owner_id	0
permission	755
type	file
address	55
...	

6

Outline

- BSD's Fast File System

- Log-structured File System

A Fast File System for UNIX

Marshall K. McKusick, William N. Joy, Samuel J. Leffler and Robert S. Fabry

Computer Systems Research Group

Why do we care about fast file system

- We want better performance!!!
- We want new features!

Let's make file systems great again!

Problems in the "old" file system

- Lots of seeks when accessing a file
  - inodes are separated from data locations
  - data blocks belong to the same file can be spread out
- Low bandwidth utilization
  - only the very last is retrieving data
  - 1 out 11 in our previous example — less than 10% if files are small
- Limited file size
- Crash recovery
- Device oblivious

What does fast file system propose?

- Cylinder groups

- Larger block sizes

- Fragments

- Allocators

- New features

  - long file names

  - file locking

  - symbolic links

  - renaming

  - quotas

What FFS proposes?

- How many of the following does FFS propose?
    - ① Cylinder groups to improve average access time
    - ② Larger block size to improve bandwidth
    - ③ Larger block size to support larger files
    - ④ Replicated superblocks for data recovery
    - ⑤ Pre-allocate blocks to improve write performance
    - A. 1
    - B. 2
    - C. 3
    - D. 4
    - E. 5

Cylinder group

- A cylinder group is a logical unit of one or several consecutive cylinders on the disk. How many of the following can a cylinder group store?
    - ① inodes
    - ② superblocks
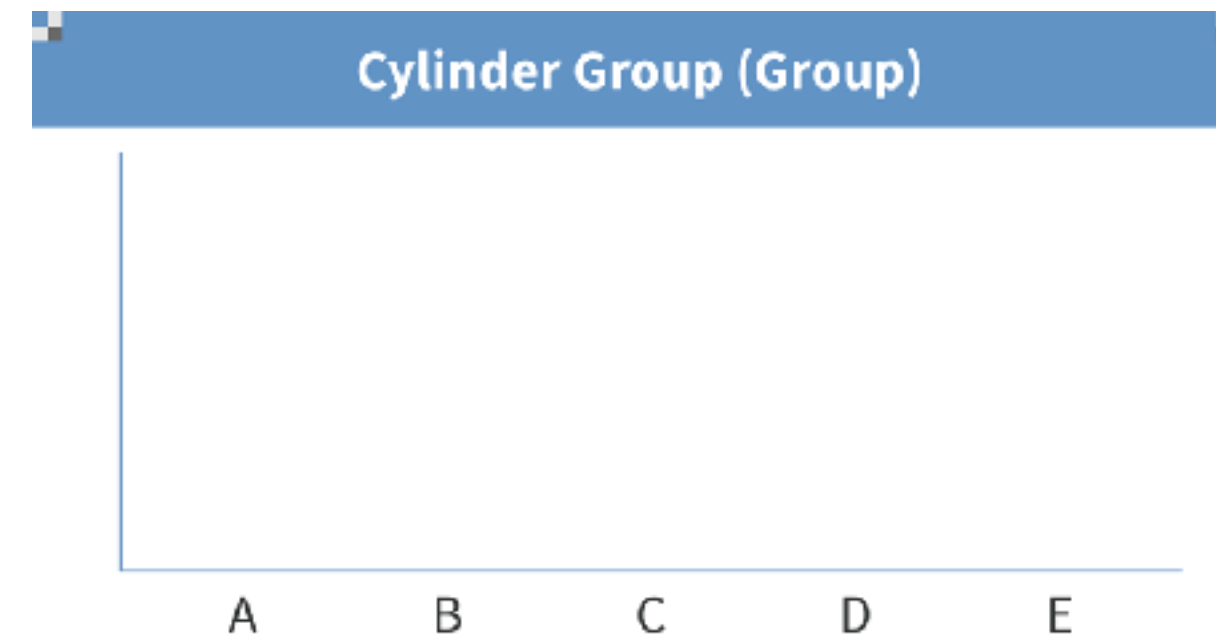   - ③ bitmap of free blocks in the cylinder group
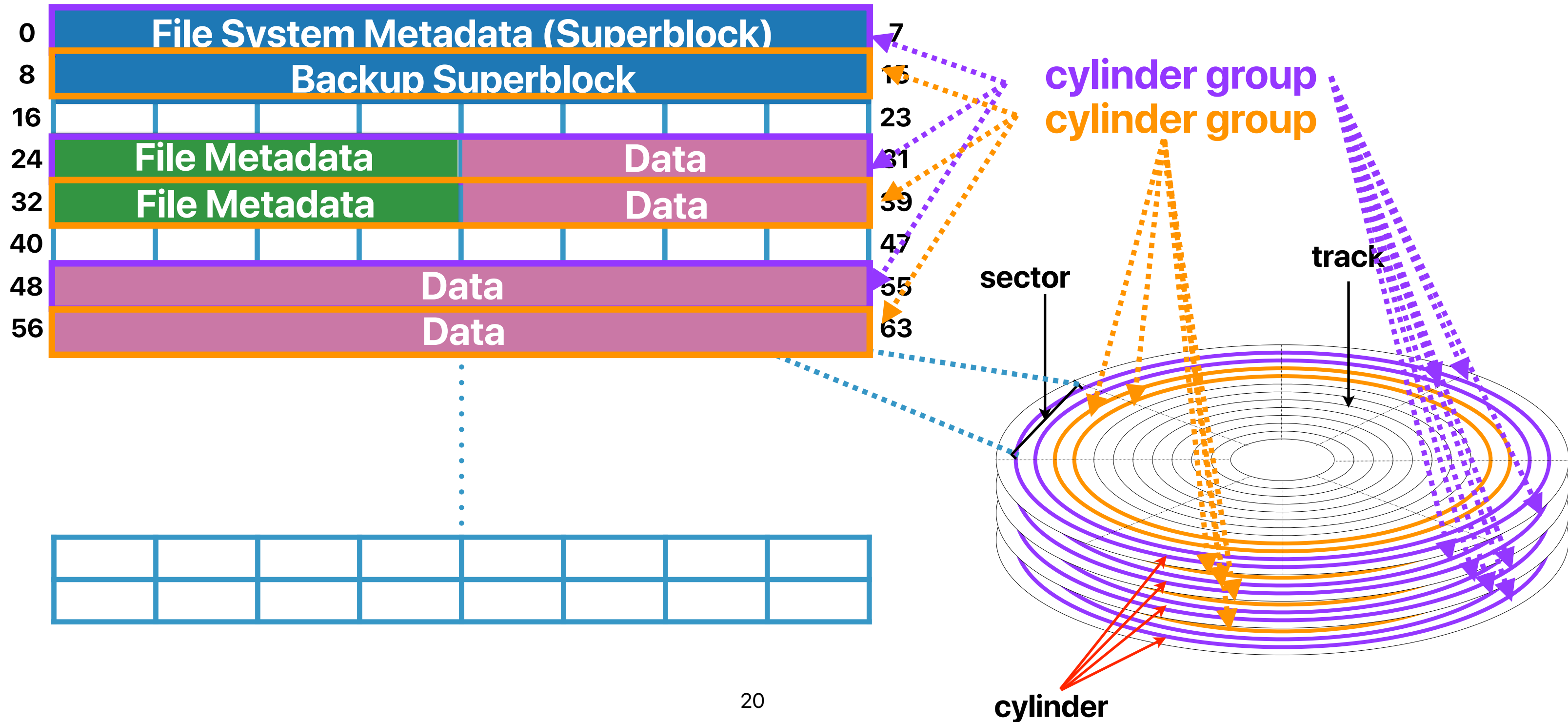   - ④ data
    - A. 0
    - B. 1
    - C. 2
    - D. 3
    - E. 4

18

Cylinder Group (Group)

A B C D E
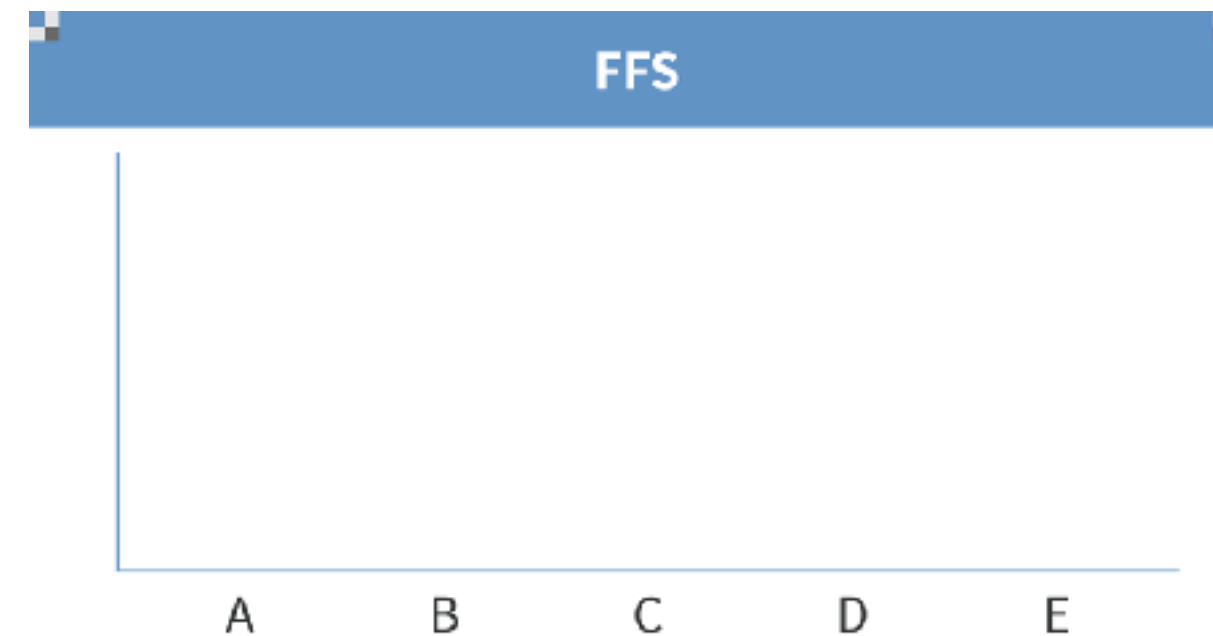
How FFS use disk blocks

Disk blocks

0 File System Metadata (Superblock)							**7**
8 Backup Superblock							**15**
16							**23**
24 File Metadata				Data			**31**
32 File Metadata				Data			**39**
40							**47**
48 Data							**55**
56 Data							**63**

cylinder group
cylinder group

track

sector

cylinder

20

Cylinder groups

- Consists of one or more consecutive cylinders on a disk
- Each cylinder group contains the following
  - redundant copy of the superblock
    - what's the benefit?
    - why not a cylinder group for all superblocks?
  - inode space
  - bitmap of free blocks within the cylinder group
  - summary of block usage
  - data
- Improves average disk access time
  - Allocating blocks within the same cylinder group for the same file
  - Placing inode along with data within the same cylinder group

Cylinder group

- A cylinder group is a logical unit of one or several consecutive cylinders on the disk. How many of the following can a cylinder group store?
    - ① inodes
    - ② superblocks
    - ③ bitmap of free blocks in the cylinder group
    - ④ data
    - A. 0
    - B. 1
    - C. 2
    - D. 3
    - E. 4

Cylinder groups

- Which of the following factor of disk access can cylinder groups help to improve when manage files?
    - A. Seek time
    - B. Rotational delay
    - C. Data transfer latency
    - D. A and B
    - E. A and C

sector

track

cylinder
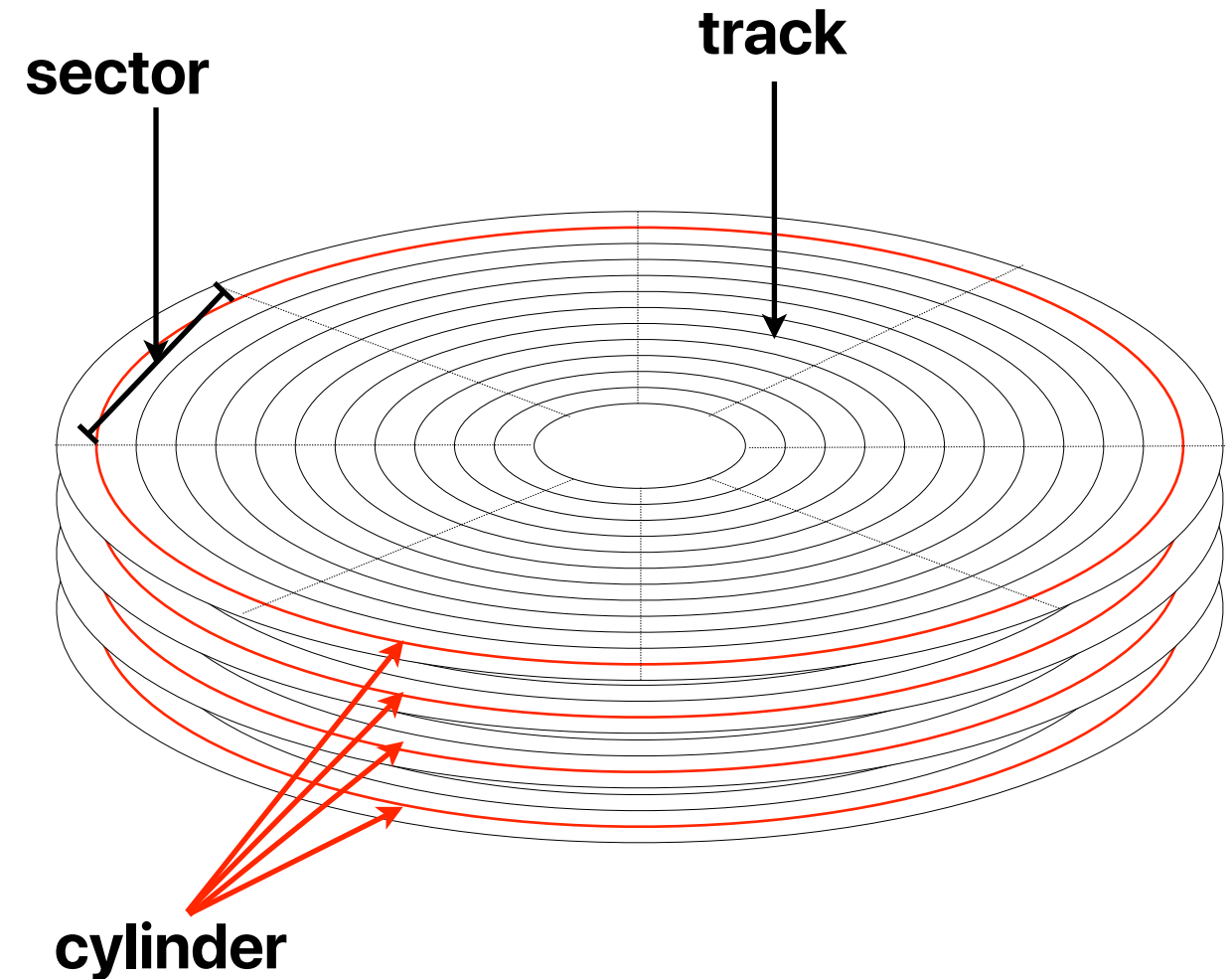
Larger block sizes

- The block size of the old file system is aligned with the block (sector) size of the disk

  - Each file can only contain a fixed number of blocks

  - Cannot fully utilize the I/O interface bandwidth

- The new file system supports larger block sizes

  - Supports larger files

  - Each I/O request can carry more data to improve bandwidth

- However, larger block size leads to internal fragments

How larger block sizes improves bandwidth

- SATA II (300MB/s in theory), 7200 R.P.M., seek time around 8 ms. Assume the controller overhead is 0.2ms. What's the bandwidth of accessing 512B sectors and 4MB consecutive sectors?

Latency = seek time + rotational delay + transfer time + controller overhead

$$= 8 \ ms + 4.17 \ ms + 13.33 \ ms + 0.2 \ ms = 25.69 \ ms$$

Bandwidth = volume_of_data over period_of_time

$$= \frac{4MB}{25.69ms} = 155.7 \ MB/sec \qquad \text{\textbf{Trading latencies with bandwidth}}$$

$$= 8 \ ms + 4.17 \ ms + 0.00167 \ us + 0.2 \ ms = 12.36 \ ms$$

$$= \frac{0.5KB}{12.36ms} = 40.45KB/sec$$

Fragments

- Addressable units within a block

- Allocates fragments from a block with free fragments if the writing file content doesn't fill up a block

Allocators

- Global allocators
  - Try to allocate inodes belong to same file together
  - Spread out directories across the disk to increase the successful rate of the previous
- Local allocators — allocate data blocks upon the request of the global allocator
  - Rotationally optimal block in the same cylinder
  - Allocate a block from the cylinder group if global allocator needs one
  - Search for blocks from other cylinder group if the current cylinder group is exhausted

What FFS proposes?

- How many of the following does FFS propose?
  - ① Cylinder groups to improve average access time
  - ② Larger block size to improve bandwidth
  - ③ Larger block size to support larger files
  - ④ Replicated superblocks for data recovery
  - ⑤ Pre-allocate blocks to improve write performance

  A. 1

  B. 2

  C. 3

  D. 4

  E. 5

  allocation [13]. This technique was not included because block allocation currently accounts for less than 10 percent of the time spent in a write system call and, once again, the current throughput rates are already limited by the speed of the available processors.

Performance of FFS

Table IIa. Reading Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Read bandwidth %		% CPU
Old 1024	750/UNIBUS	29	29/983	3	11
New 4096/1024	750/UNIBUS	221	221/983	22	43
New 8192/1024	750/UNIBUS	233	233/983	24	29
New 4096/1024	750/MASSBUS	466	466/983	47	73
New 8192/1024	750/MASSBUS	466	466/983	47	54

not the case for old FS

writes in FFS are slower than reads

Table IIb. Writing Rates of the Old and New UNIX File Systems

Type of file system	Processor and bus measured	Speed (Kbytes/s)	Write bandwidth %		% CPU
Old 1024	750/UNIBUS	48	48/983	5	29
New 4096/1024	750/UNIBUS	142	142/983	14	43
New 8192/1024	750/UNIBUS	215	215/983	22	46
New 4096/1024	750/MASSBUS	323	323/983	33	94
New 8192/1024	750/MASSBUS	466	466/983	47	95

CPU load is fine given that UFS is way too slow!

37

How is BSD FFS doing?

- Regarding the performance of BSD FFS, please identify how many of the following statements is/are true
    - ① BSD FFS is performing better than UFS regardless of reads and writes
    - ② The performance of reading data is faster than writing data in BSD FFS, while the reading is slower than writing in UFS
    - ③ The bandwidth utilization of BSD FFS is better than UFS
    - ④ The CPU utilization of BSD FFS is higher than UFS
    - A. 0
    - B. 1
    - C. 2
    - D. 3
    - E. 4

Writes

- Larger overheads than the old file system as the new file system allocates blocks after write requests occur — Why not optimize for writes?
  - 10% of overall time
  - writes are a lot faster already
- Writing metadata is synchronous rather than asynchronous — What's the benefit of synchronous writes?
  - Consistency

What does fast file system propose?

- Cylinder groups — **improve spread-out data locations**
- Larger block sizes — **improve bandwidth and file sizes**
- Fragments — **improve low space utilization due to large blocks**
- Allocators — **address device oblivious**
- New features
  - long file names
  - file locking
  - symbolic links
  - renaming
  - quotas

The design and implementation of a log-structured file system

Mendel Rosenbaum and John K. Ousterhout
Univ. of California, Berkeley

Why LFS?

- How many of the following problems is/are Log-structured file systems trying address?
    - ① ✔ The performance of small random writes
    - ② The efficiency of large file accesses
    - ③ The space overhead of metadata in the file system
    - ④ Reduce the main memory space used by the file system
    - A. 0
    - B. 1
    - C. 2
    - D. 3
    - E. 4

In designing a log-structured file system we decided to focus on the efficiency of small-file accesses, and leave it

The notion of logging is not new, and a number of recent file systems have incorporated a log as an auxiliary structure to speed up writes and crash recovery[2, 3]. How-

Why LFS?

- Writes will dominate the traffic between main memory and disks — Unix FFS is designed under the assumption that only 10% of the traffic are writes

  - Who is wrong? **UFS is published in 1984**

  - As system memory grows, frequently read data can be cached efficiently

  - Every modern OS aggressively caches — use "free" in Linux to check

- Gaps between sequential access and random access

- Conventional file systems are not RAID aware

47

Why LFS?

- How many of the following problems is/are Log-structured file systems trying address?
    - ① ✔ The performance of small random writes
    - ② The efficiency of large file accesses
    - ③ The space overhead of metadata in the file system
    - ④ Reduce the main memory space used by the file system
    - A. 0
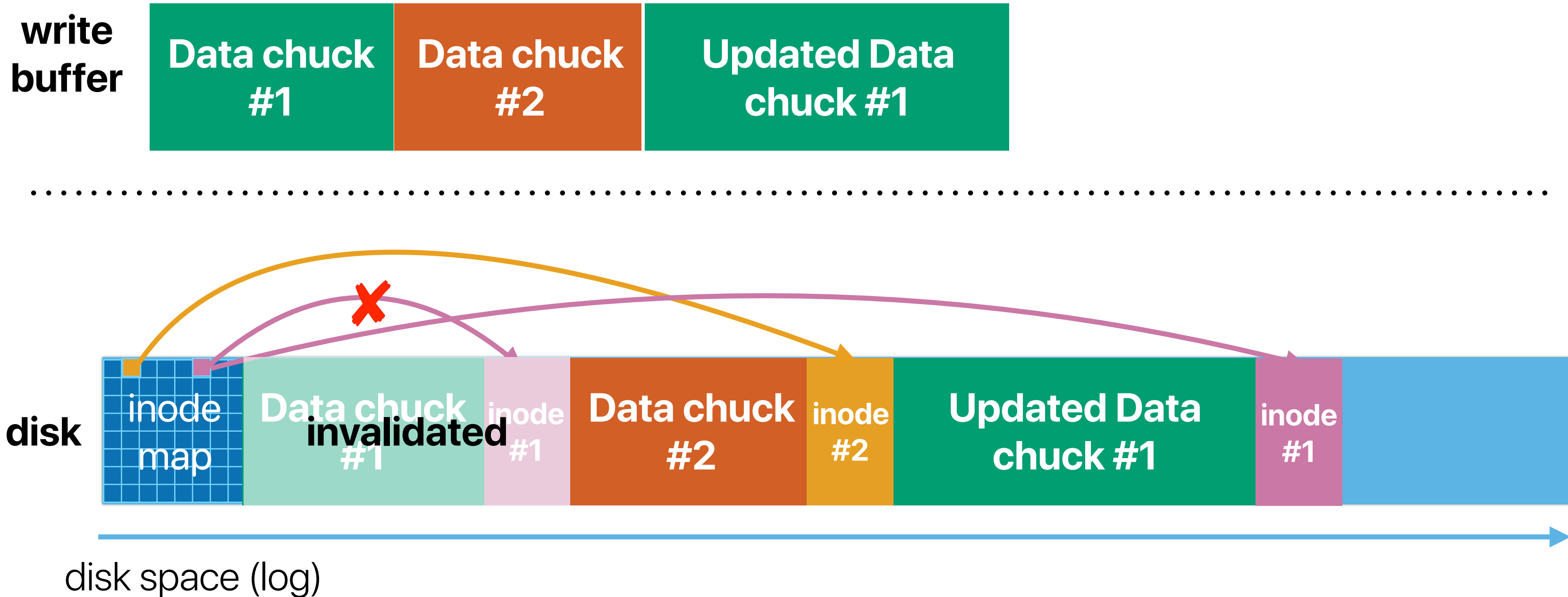   - B. 1
    - C. 2
    - D. 3
    - E. 4

Problems with BSD FFS

- Data are spread out the whole disk
    - Can achieve sequential access within each file, but the distance between files can be far
    - An inode needs a standalone I/O in addition to file content
    - Creating files take at least five I/Os with seeks — can only use 5% bandwidth for data
        - 2 for file attributes
            - You have to check if the file exists or not
            - You have to update after creating the file
        - 1 for file data
        - 1 for directory data
        - 1 for directory attributes
- Writes to metadata are synchronous
    - Good for crash recovery, bad for performance

What does LFS propose?

- Buffering changes in the system main memory and commit those changes sequentially to the disk with fewest amount of write operations

LFS in motion

write buffer

| Data chuck #1 | Data chuck #2 | Updated Data chuck #1 |

disk

inode map | Data chuck **invalidated** #1 | inode #1 | Data chuck #2 | inode #2 | Updated Data chuck #1 | inode #1

disk space (log)

Why LFS?

- How many of the following problems is/are Log-structured file systems trying address?

  ① ✓ The performance of small random writes

  ② The efficiency of large file accesses **leave it for the hardware designer**

  ③ The space overhead of metadata in the file system
  increases due to garbage collection and inode maps

  ④ Reduce the main memory space used by the file system
  increases due to write caching

  A. 0
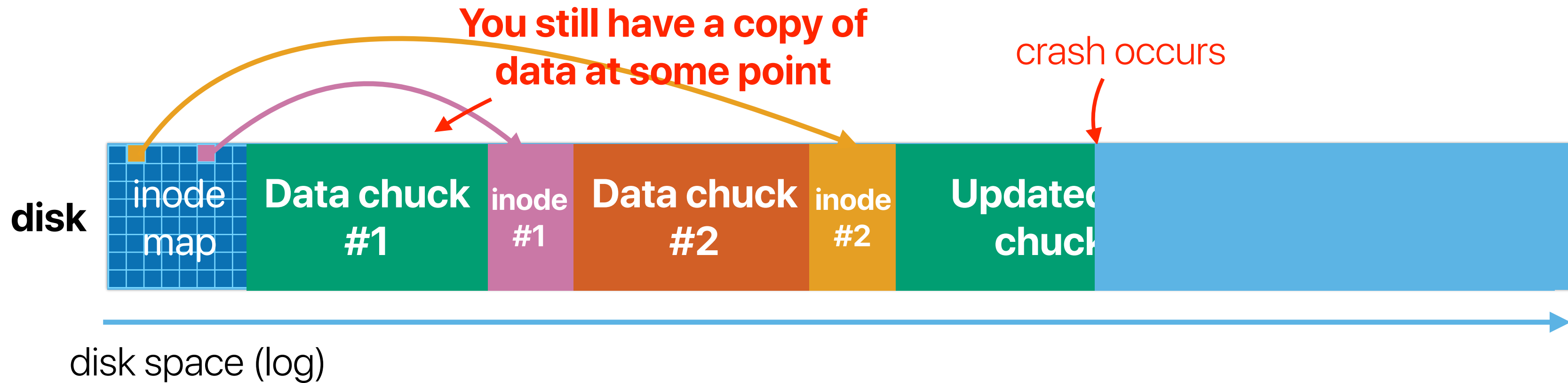
  B. 1

  C. 2

  D. 3

  E. 4

> In designing a log-structured file system we decided to focus on the efficiency of small-file accesses, and leave it to hardware designers to improve bandwidth for large-file accesses. Fortunately, the techniques used in Sprite LFS work well for large files as well as small ones.
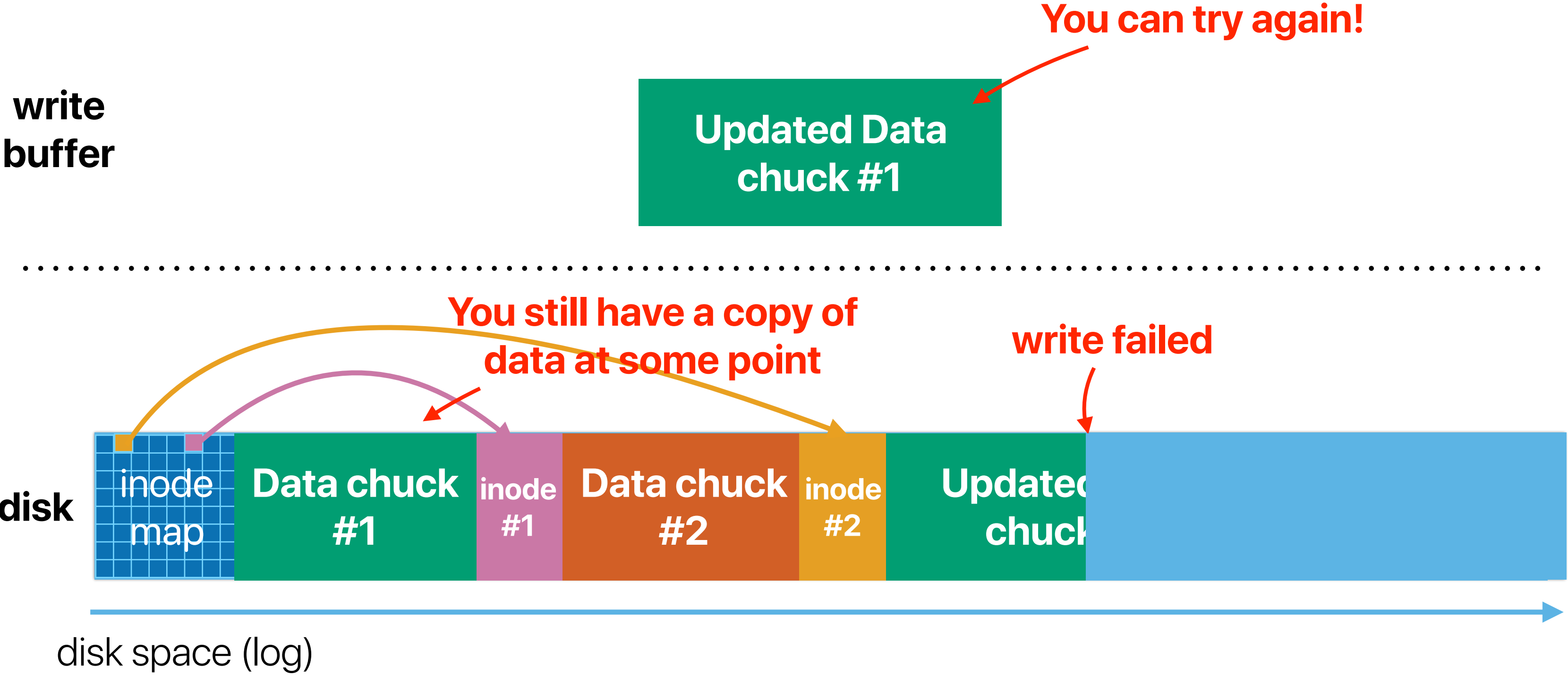
Crash recovery

- Checkpointing
    - Create a redundant copy of important file system metadata periodically

- Roll-forward
    - Scan through/replay the log after checkpointing

LFS v.s. crash

write buffer

disk

You still have a copy of data at some point

crash occurs

| inode map | Data chuck #1 | inode #1 | Data chuck #2 | inode #2 | Updated chuck | |

disk space (log)

54

LFS v.s. write failed

Segment cleaning/Garbage collection

- Reclaim invalidated segments in the log once the latest updates are checkpointed

- Rearrange the data allocation to make continuous segments

- Must reserve enough space on the disk

  - Otherwise, every writes will trigger garbage collection

  - Sink the write performance

Announcement

- Midterm due next Tuesday midnight
  - Up after this lecture
  - Unlimited time, but only one shot
  - Don't cheat
- Reading quiz due next Thursday
- Project due 3/3
  - We highly recommend you to fresh install a Ubuntu 16.04.6 Desktop version within a VirtualBox
    - Virtual box is free
    - If you crash the kernel, just terminate the instance and restart virtual box
  - Use office hours to discuss projects