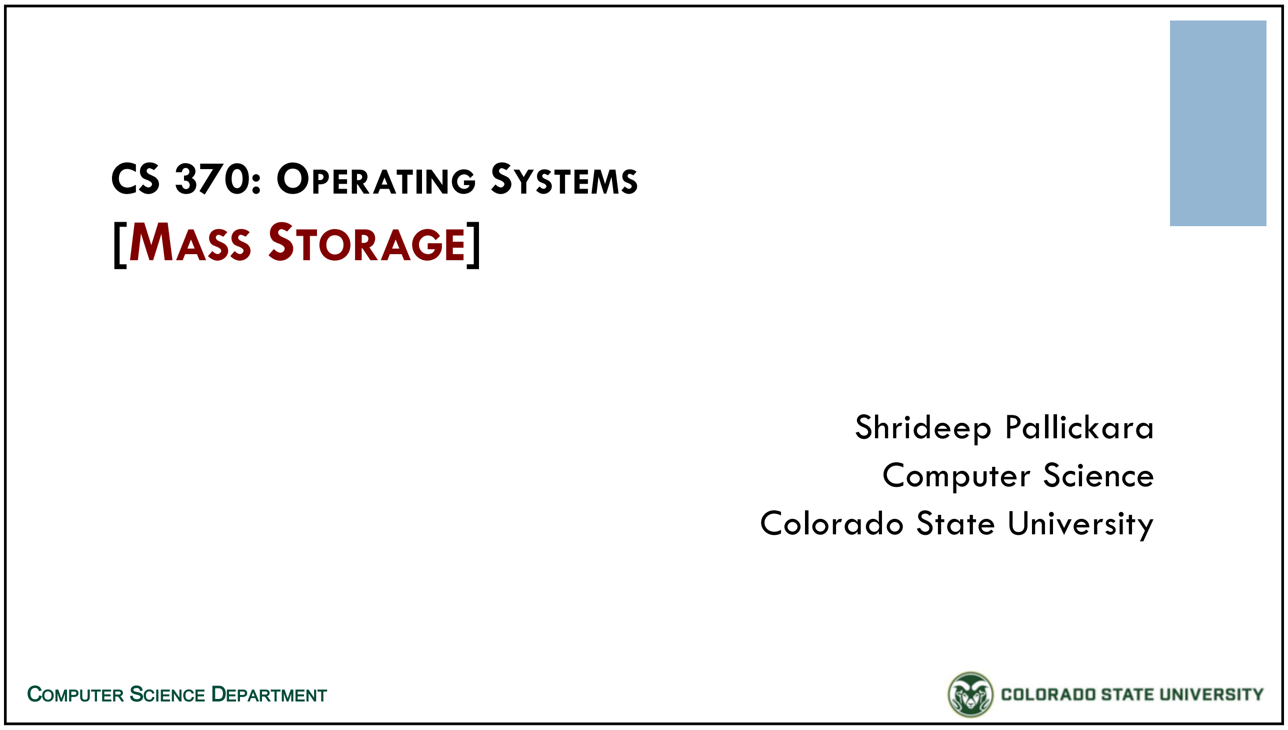


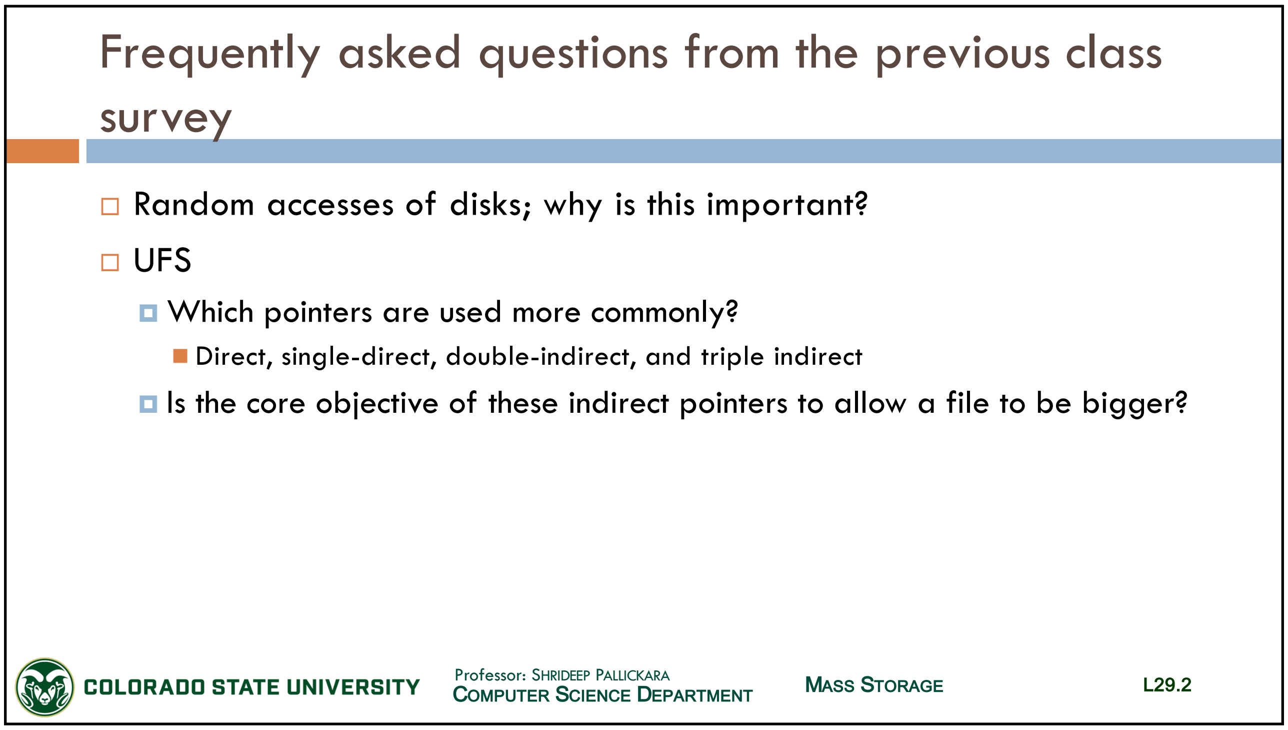
CS 370: Operating Systems
[Mass Storage]

Shrideep Pallickara
Computer Science
Colorado State University

Computer Science Department

Colorado State University

1



Frequently asked questions from the previous class survey

- Random accesses of disks; why is this important?
- UFS
  - Which pointers are used more commonly?
    - Direct, single-direct, double-indirect, and triple indirect
  - Is the core objective of these indirect pointers to allow a file to be bigger?

Colorado State University | Professor: Shrideep Pallickara Computer Science Department | Mass Storage | L29.2

2

Topics covered in this lecture

- Wrap-up of File Systems
  - NTFS
- Flash Memory
- RAID

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.3

3

WINDOWS NEW TECHNOLOGY FILE SYSTEM (NTFS)

COMPUTER SCIENCE DEPARTMENT

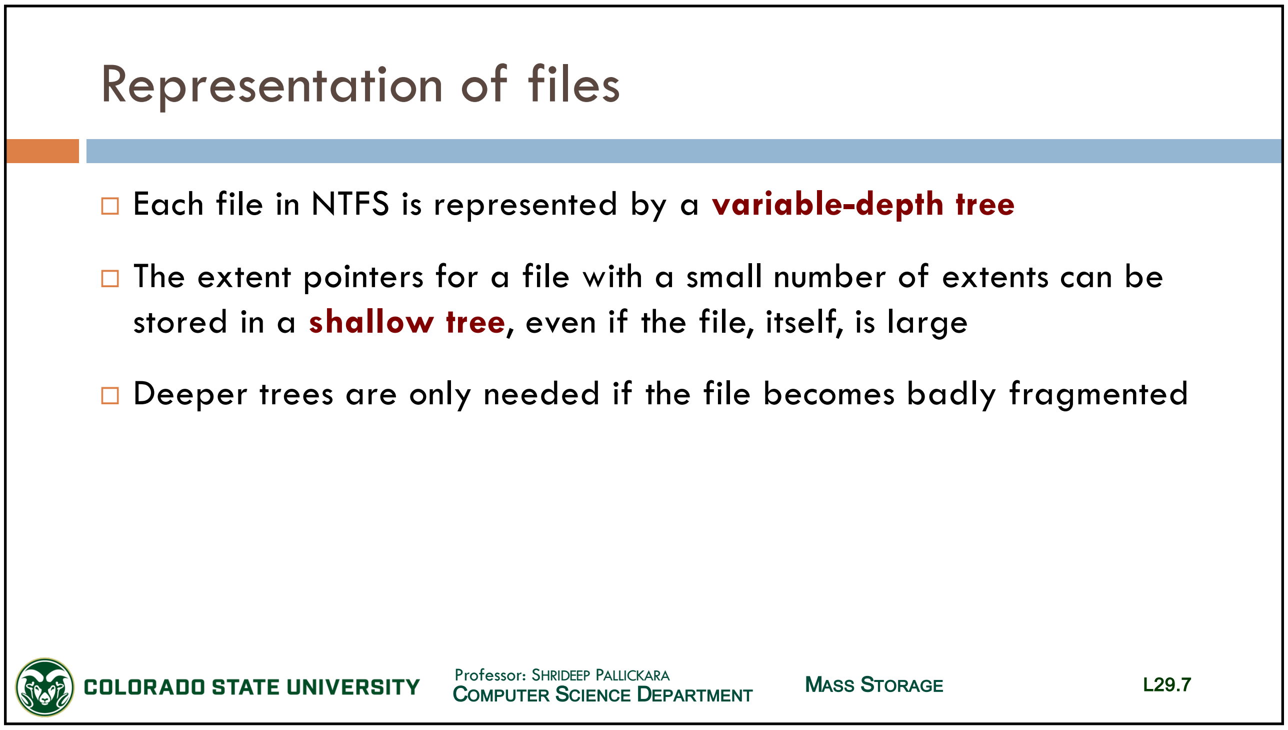
4

New Technology File System (NTFS)

- NTFS was first released in 1993
- Improved on Microsoft's FAT file system with many new features
  - Including new index structures to improve performance, more flexible file metadata, improved security, and reliability

NTFS: Extents and flexible trees

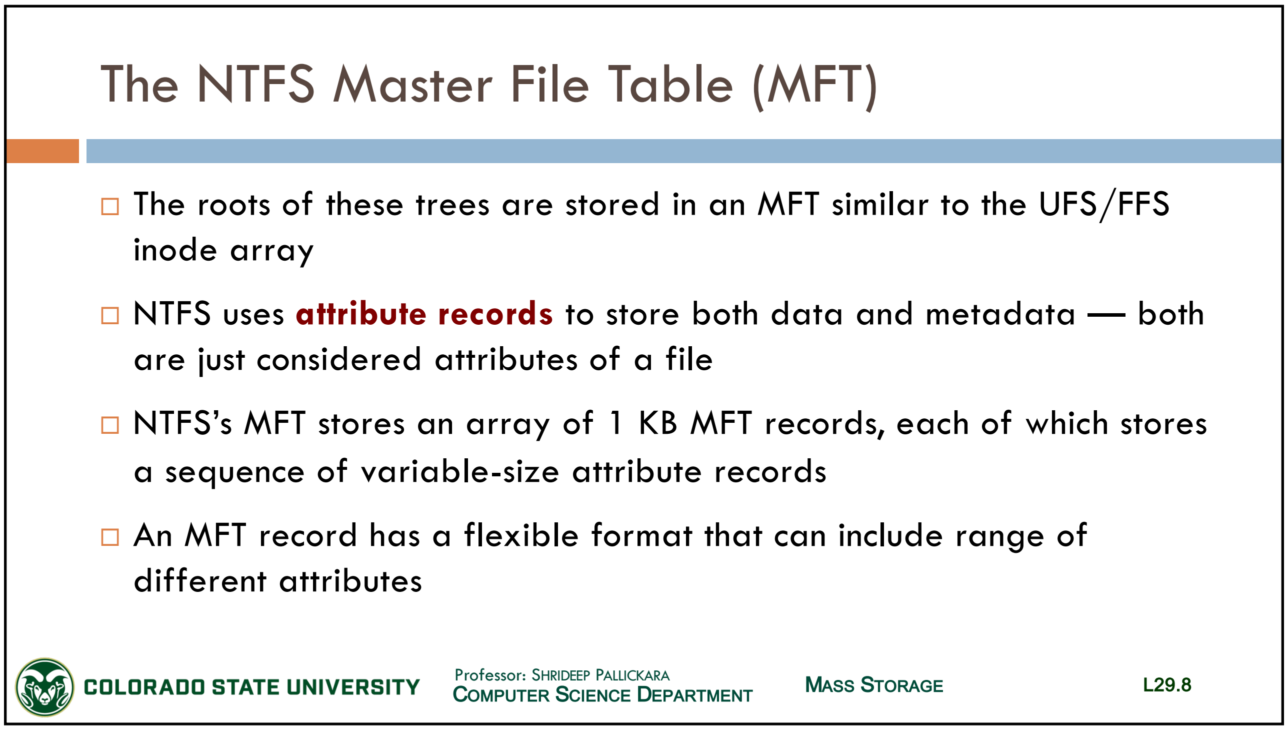
- Rather than tracking individual file blocks, NTFS tracks **extents**
  - Variable-sized regions of files that are each stored in a contiguous region on the storage device
- Whereas UFS tracks file blocks with a fixed tree, NTFS tracks extents with flexible trees
  - Many other recent file systems such as Linux ext4 and btrfs now leverage this idea as well



Representation of files

- Each file in NTFS is represented by a **variable-depth tree**
- The extent pointers for a file with a small number of extents can be stored in a **shallow tree**, even if the file, itself, is large
- Deeper trees are only needed if the file becomes badly fragmented

7



The NTFS Master File Table (MFT)

- The roots of these trees are stored in an MFT similar to the UFS/FFS inode array
- NTFS uses **attribute records** to store both data and metadata — both are just considered attributes of a file
- NTFS's MFT stores an array of 1 KB MFT records, each of which stores a sequence of variable-size attribute records
- An MFT record has a flexible format that can include range of different attributes

8

In addition to data attributes, an MFT record has three common metadata attribute types

- **Standard information**
  - Needed for all files and includes the file's creation time, modification time, access time, owner ID, and security specifier
  - Also includes a set of flags indicating basic information like whether the file is a read only file, a hidden file, or a system file
- **File name**
  - Holds the file's name and the file number of its parent directory
  - Because a file can have multiple names (e.g., if there are multiple hard links to the file), it may have multiple file name attributes in its MFT record

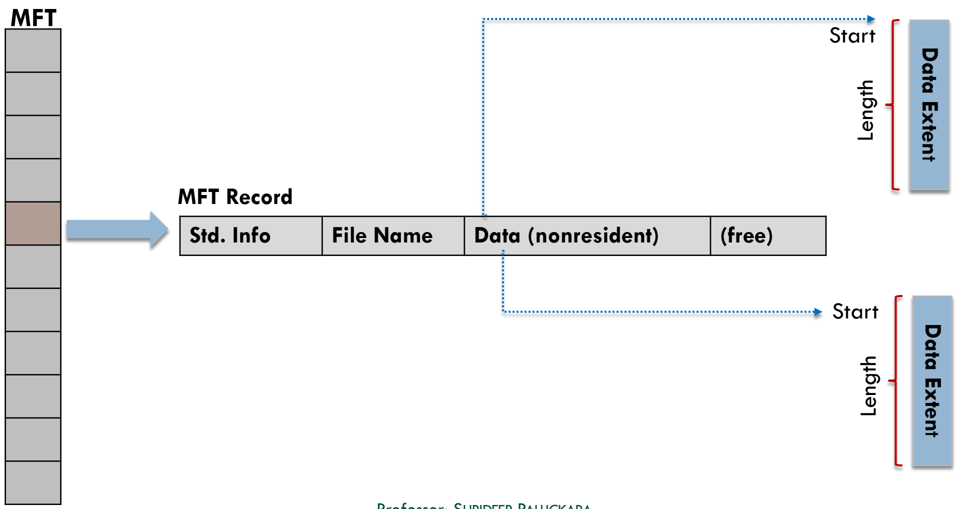
Attributes

- A file's metadata may include a *variable number* of variable sized attributes
- Thus, a file's metadata may be larger than a single MFT record can hold

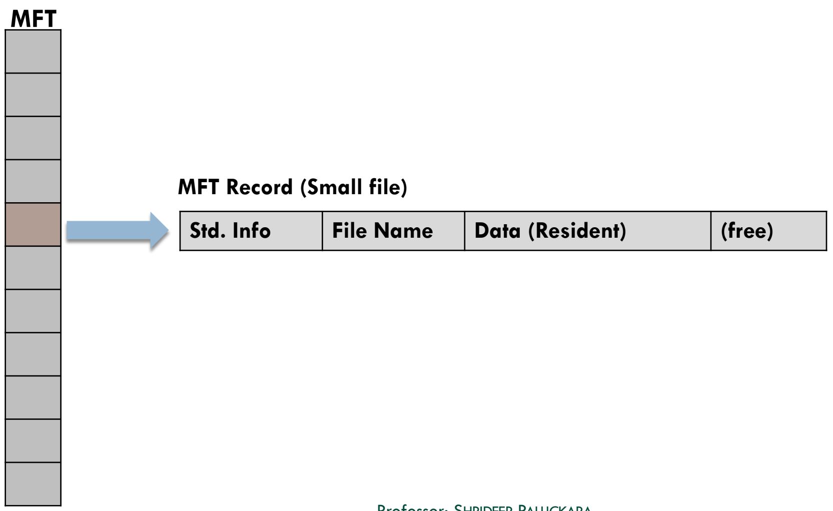
Resident and non-resident attribute records

- Some attributes can be too large to fit in an MFT record (e.g., data extents)
  - While some can be small enough to fit (e.g., a file's last modified time)
- An attribute can therefore be **resident** or **non-resident**
  - A resident attribute stores its contents directly in the MFT record
  - A non-resident attribute stores extent pointers in its MFT record and stores its contents in those extents

NTFS index structure for basic file with 2 data extents



A small file's data can be resident, meaning the file's data is stored in the MFT record



Metadata files

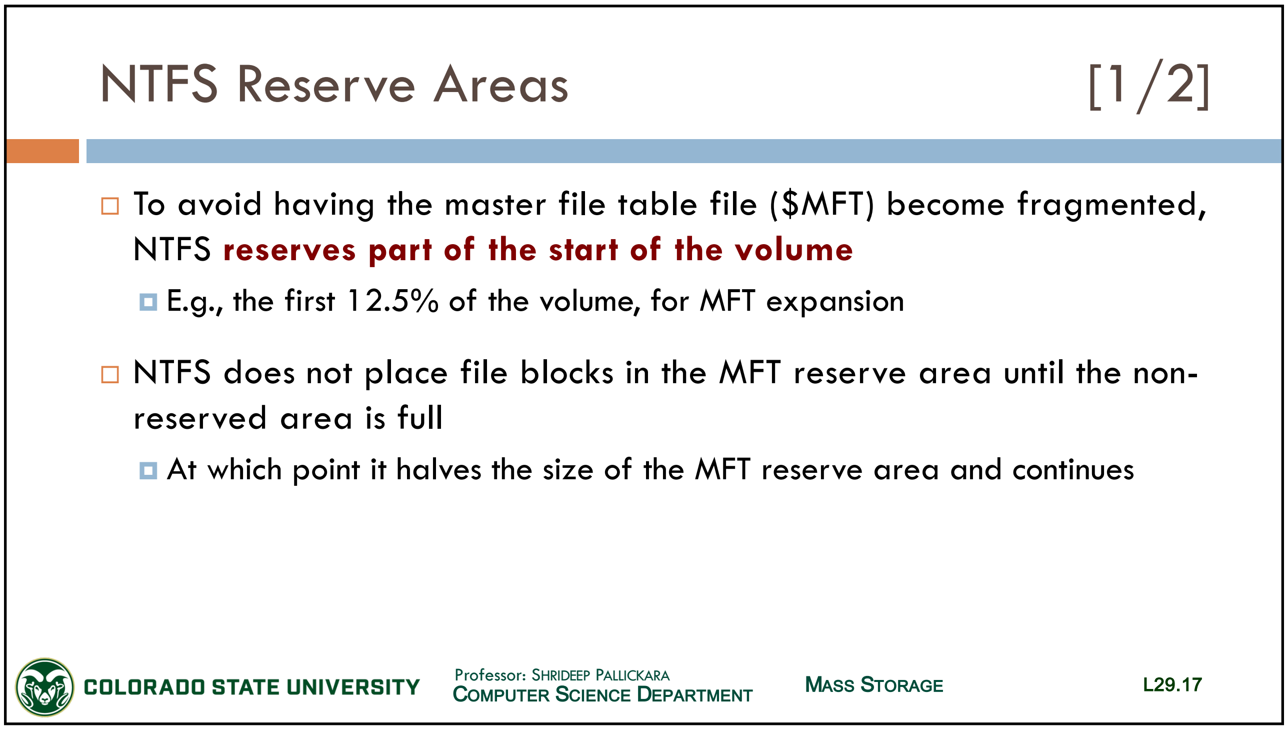
- NTFS stores almost all of its metadata in about a dozen ordinary files with well-known low-numbered file numbers
  - File number 2, log of metadata changes
  - File number 3 (volume), 4 (attribute defs)
  - File number 5 is the root directory
  - File number 6 is the free space bitmap, 7 (volume boot record)
  - File number 8 contains a list of the volume's bad blocks
  - File number 9, called `$Secure`, contains security and access control info
  - File number 10 (Unicode upper case table), 11 (optional extensions)

How is the MFT stored? [1/2]

- Even the master file table, itself, is stored as a file, **file number 0**, called **$MFT**
  - File number 1 is a **mirror** of the $MFT
- So, we need to find the first entry of the MFT in order to read the MFT!
- To locate the MFT, the **first sector of an NTFS volume** includes a pointer to the first entry of the MFT

How is the MFT stored? [2/2]

- Storing the MFT in a file avoids the need to statically allocate all MFT entries as a fixed array in a predetermined location
- Instead, NTFS starts with a small MFT and grows it as new files are created and new entries are needed

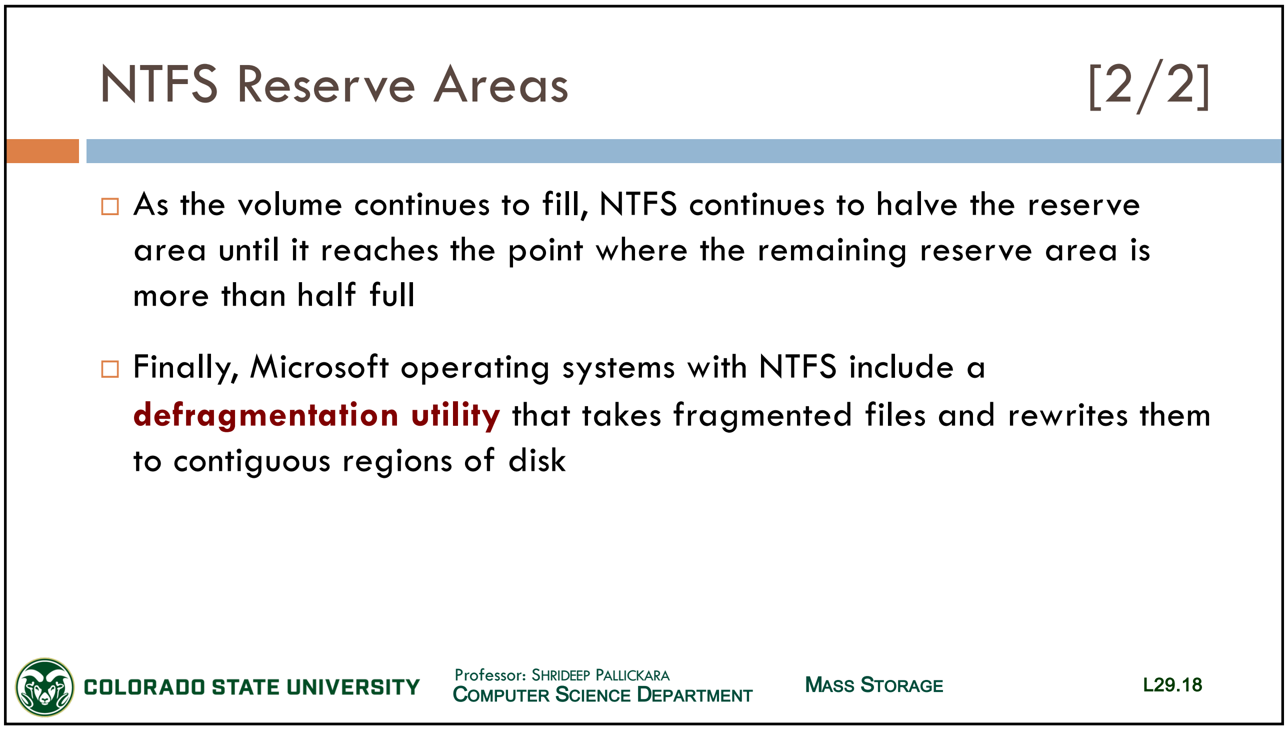


NTFS Reserve Areas [1/2]

- To avoid having the master file table file ($MFT) become fragmented, NTFS **reserves part of the start of the volume**
  - E.g., the first 12.5% of the volume, for MFT expansion
- NTFS does not place file blocks in the MFT reserve area until the non-reserved area is full
  - At which point it halves the size of the MFT reserve area and continues

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.17

17



NTFS Reserve Areas [2/2]

- As the volume continues to fill, NTFS continues to halve the reserve area until it reaches the point where the remaining reserve area is more than half full
- Finally, Microsoft operating systems with NTFS include a **defragmentation utility** that takes fragmented files and rewrites them to contiguous regions of disk

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.18

18

NTFS: Locality heuristics

- Rather than trying to keep the allocation bitmap for the entire disk in memory
  - The system caches the allocation status for a smaller region of the disk and searches that region first

19

FLASH MEMORY

20

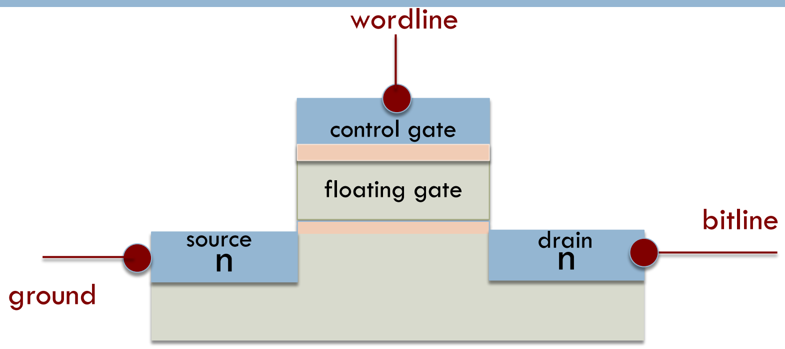
Flash memory is a type of a solid-state storage

- No moving parts … and stores data using electrical circuits
  - Can have better random I/O performance than HDDs, use less power, and is less vulnerable to physical damage
  - But significantly more expensive per byte

Transistors

- It takes one transistor to store a bit
- Ordinary transistors are electronic switches
  - Turned on and off by electricity
- Strength: Computer can store information simply by **passing patterns of electricity** through its memory circuits
- Weakness: As soon as power is turned off, transistors revert to their original state (loses all information)
  - Electronic amnesia!

Transistors in flash memory



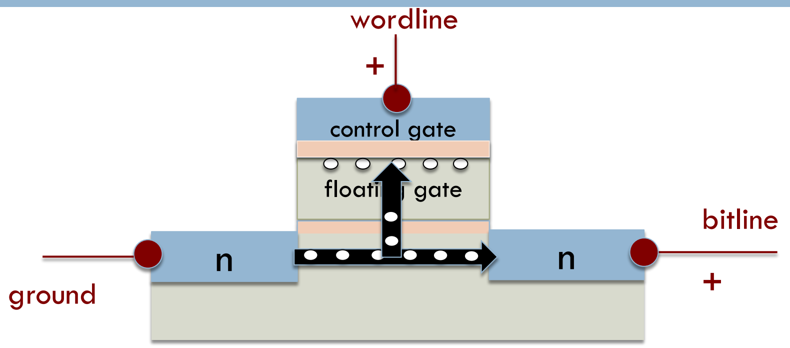
The source and drain regions are rich in electrons (n-type silicon)

Electrons cannot flow from source to drain, because of the electron-deficient p-type material between them

A gate that floats?

- The extra gate in our transistor "floats" — it is not connected to any circuit
- Since the floating gate is <u>entirely surrounded</u> by an **insulator**, it will hold an electrical charge for months or years without requiring any power
- Even though the floating gate is not electrically connected to anything, it can be charged or discharged
  - Via **electron tunneling** by running a sufficiently high-voltage current near it

Transistors in flash memory



The presence of electrons on the floating gate is how a flash transistor stores a **one**

Electrons stay there indefinitely, even when positive voltages are removed AND whether power is supplied to the unit or not

Electrons can be flushed out by putting a negative voltage on the wordline. REPELS electrons back.

How do you read what is stored in the floating gate?

- The floating gate's state of charge affects the transistor's threshold voltage for activation
- State can be detected by applying an **intermediate voltage** to the transistor's control gate
  - Intermediate voltage will only be sufficient to activate the transistor if the floating gate is charged

Flash storage: Erasure blocks

- Before flash memory can be written, it must be *erased* by setting each cell to a logical "1"
- Can only be erased in large units called **erasure blocks** (128-512 KB)
- Slow operation: takes several milliseconds
- Erasing an erasure block is what gives "flash memory" its name …
  - Resemblance to the flash of a camera

Write page and read page

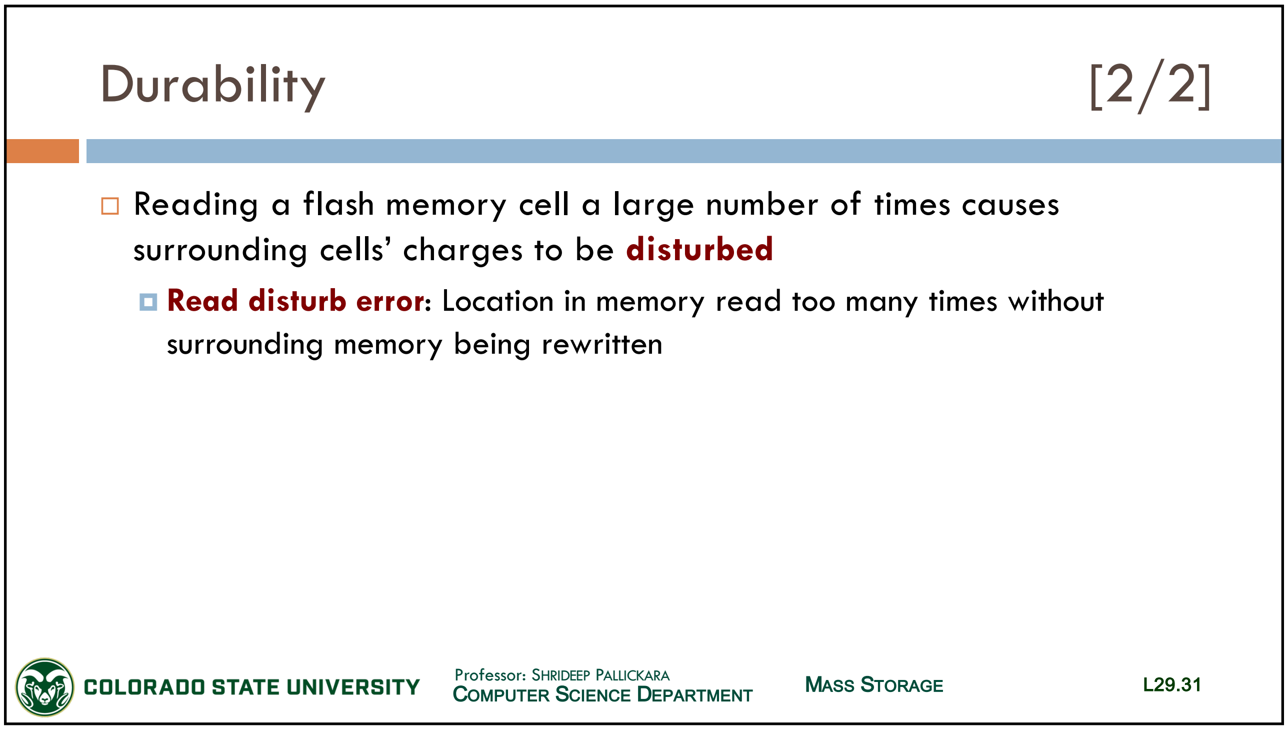
- Write Page:
  - Once erased, flash memory can be written on a page-by-page basis
  - Each page is typically 2-4 KB
  - Writing a page takes about 10s of microseconds
- Read page
  - Flash memory is read on a page-by-page basis
  - Reading a page takes about 10s of microseconds

Challenges in writing to a page

- To write a page, it's **entire erasure block** must first be erased
  - Erasure is slow and affects a large number of pages
- Flash translation layer (FTL)
  - Maps logical flash pages to different physical pages on the flash device
  - When logical page is overwritten, the FTL writes the new version to a free, already-erased physical page
    - … and remaps logical page to that physical page
  - Write remapping significantly improves performance

Durability [1/2]

- Normally, flash memory can retain state for months or years without power
- However, **high current loads** from flashing and writing memory *causes circuits to degrade*
  - After a few 1000~1,000,000 erase cycles **a cell may wear out** … cannot reliably store a bit

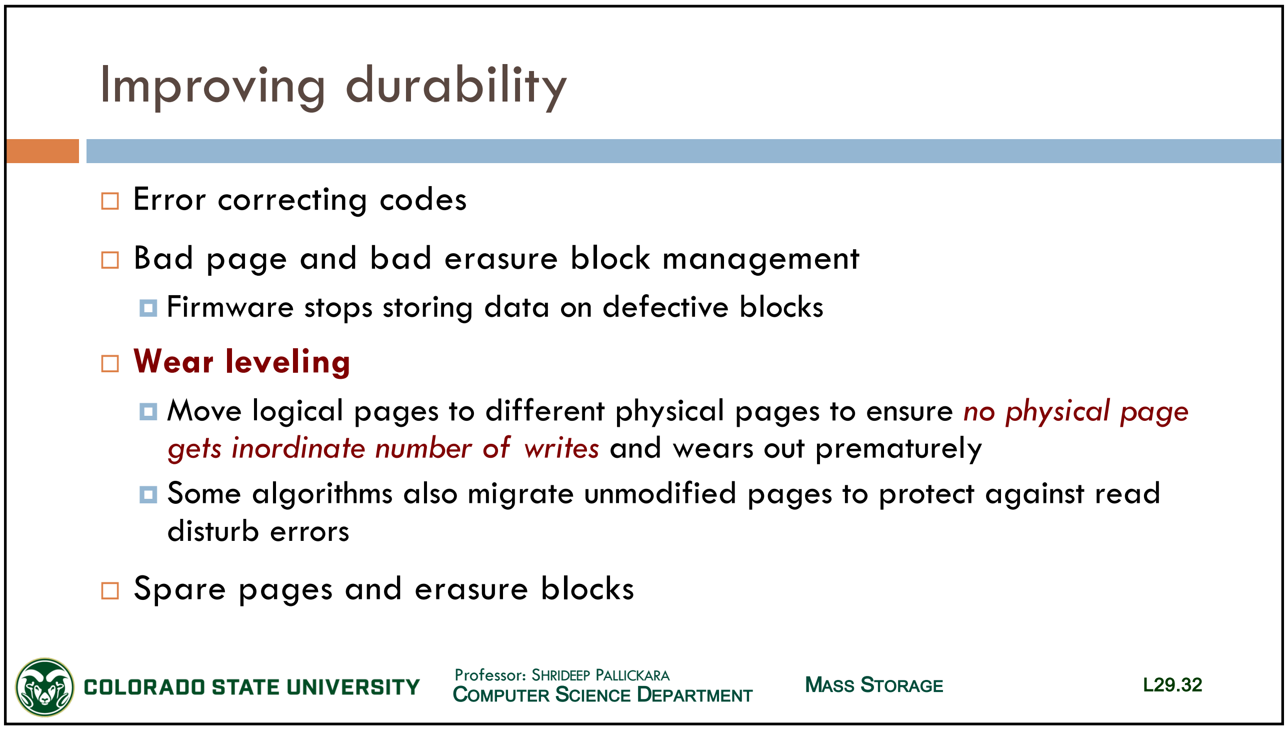


Durability [2/2]

- Reading a flash memory cell a large number of times causes surrounding cells' charges to be **disturbed**
  - **Read disturb error**: Location in memory read too many times without surrounding memory being rewritten

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.31

31



Improving durability

- Error correcting codes
- Bad page and bad erasure block management
  - Firmware stops storing data on defective blocks
- **Wear leveling**
  - Move logical pages to different physical pages to ensure *no physical page gets inordinate number of writes* and wears out prematurely
  - Some algorithms also migrate unmodified pages to protect against read disturb errors
- Spare pages and erasure blocks

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.32

32

Parameters for the Intel 710 Series SSD

- Capacity 300 GB
- Page Size 4 KB

- Performance
  - Bandwidth (Sequential Reads) 270 MB/s
  - Bandwidth (Sequential Writes) 210 MB/s
  - Read/ Write Latency 75 µs
  - Random Reads Per Second 38,500
  - Random Writes Per Second 2,000
    - 2,400 with 20% space reserve

Parameters for the Intel 710 Series SSD

- Interface SATA 3 Gb/ s

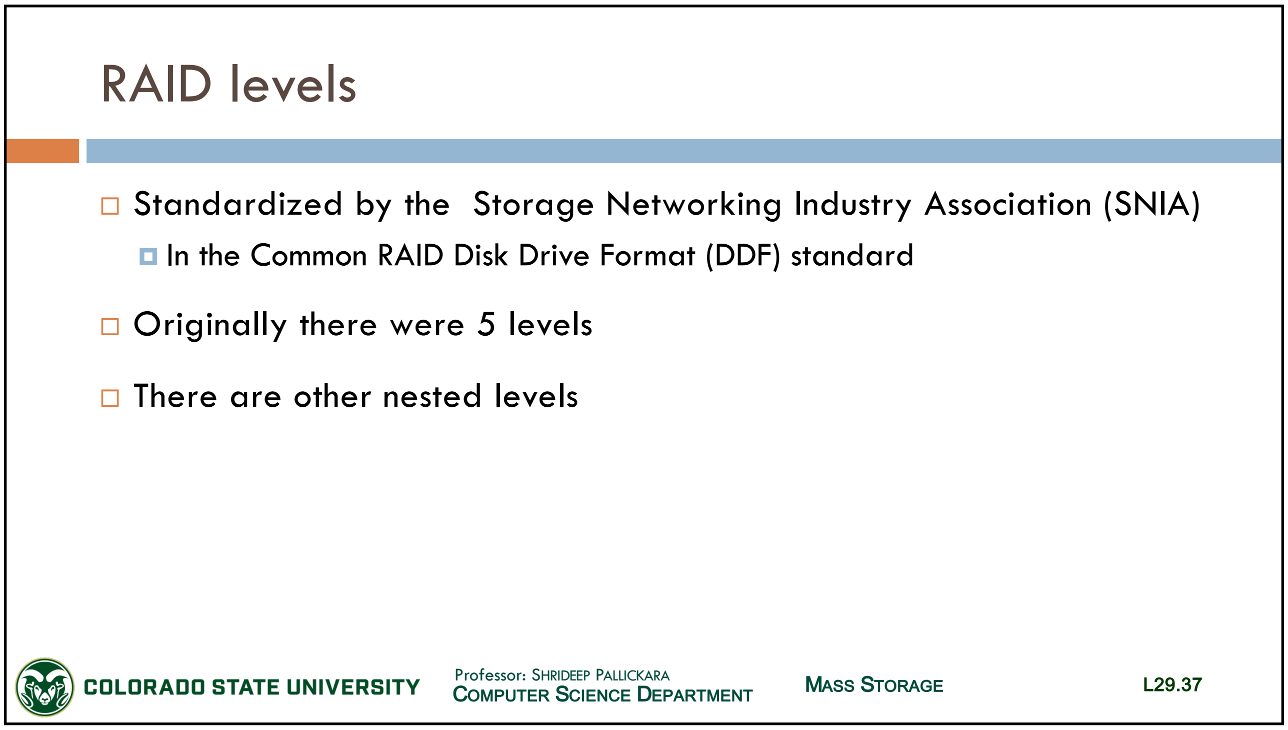
- Endurance
  - Endurance 1.1 PB
  - 1.5 PB with 20% space reserve

- Power
  - Power Consumption Active/ Idle 3.7 W / 0.7 W

RAID STRUCTURE

RAID involves using large number of disks in parallel

- Improves **rate** at which data be can read/written
- Increases **reliability** of storage
  - Redundant information can be stored on multiple disks
  - Failure of 1 disk should not result in loss of data
- Redundant Array of ~~Inexpensive~~ **Independent** Disks

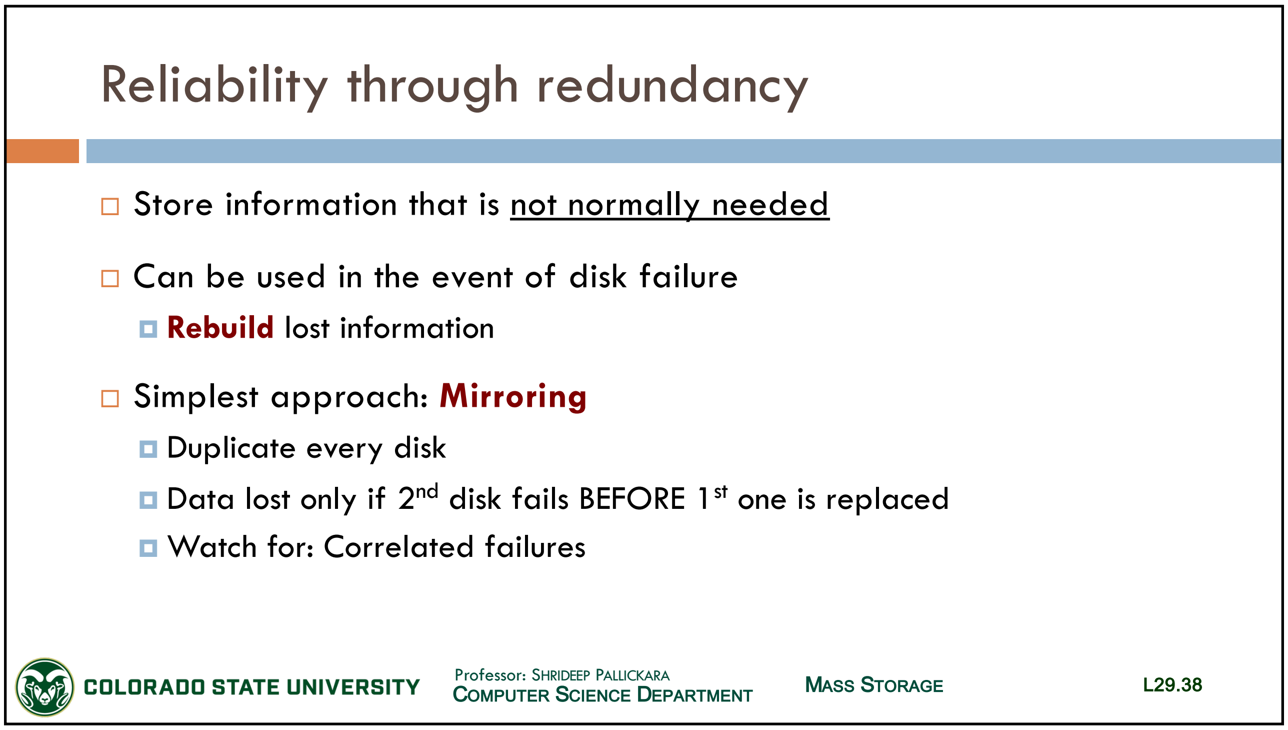


RAID levels

- Standardized by the Storage Networking Industry Association (SNIA)
  - In the Common RAID Disk Drive Format (DDF) standard
- Originally there were 5 levels
- There are other nested levels

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.37

37



Reliability through redundancy

- Store information that is <u>not normally needed</u>
- Can be used in the event of disk failure
  - **Rebuild** lost information
- Simplest approach: **Mirroring**
  - Duplicate every disk
  - Data lost only if 2nd disk fails BEFORE 1st one is replaced
  - Watch for: Correlated failures

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT MASS STORAGE L29.38

38

RAID parallelism

- **Stripe** data across disks
- Objectives
  1. Increase throughput
  2. Reduce response times of large accesses

RAID Parallelism: Stripe data across disks **Bit level** striping

- **Split bits** of each byte across multiple disks
  - 8 disks: Bit **`i`** of each byte written to disk **`i`**
  - Bit 3 written to disk 3
- Array of 8 disks treated as a single disk
  - 8 times the access rate
  - Every disk participates in every read/write

RAID Parallelism: Block-level striping

- **Blocks** of a file are striped across multiple disks
- When there are **`n`** disks
  - Block **`i`** of the file written to …
  - Disk: **`(i mod n) + 1`**
    - 4 disks: Block 8 of file goes to disk 1
    - 4 disks: Block 9 of file goes to disk 2

RAID levels

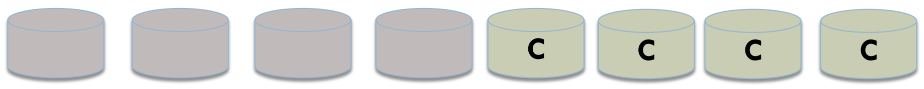
- Striping improves transfer rates
  - BUT not reliability
- Disk striping usually combined with **parity**
- Different schemes classified according to levels
  - RAID levels

RAID 0: Stripe blocks without redundancy

- No mirroring
- No parity

43

RAID 1: Disk mirroring



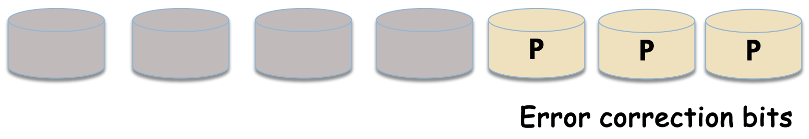
- Each disk is mirrored

44

RAID 2: Memory style error correcting code

- **Parity bit** records number of 1 bits in byte
  - Even: parity 0
  - Odd: parity 1
- Use to detect single-bit errors
- Error correcting schemes
  - 2 or more extra bits to recover from single-bit errors

RAID 2: Error Correcting Codes

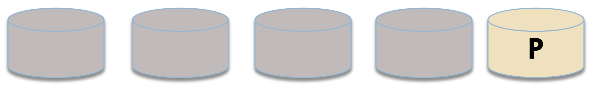


- **If one disk fails:**
- Remaining bits of the byte **+** error correction bits
  - Read from other disks
- **Reconstruct** damaged data

RAID 3: Single parity bit used for error correction

- We can identify damaged sector
- Figure out if any bit in sector is 0 or 1
  - Compute parity of corresponding bits from other sectors
    - If parity of remaining bits == stored parity
      - Missing bit = 0
    - Otherwise, missing bit = 1

RAID 3: Single parity bit used for error correction



Error correction bits

Issues

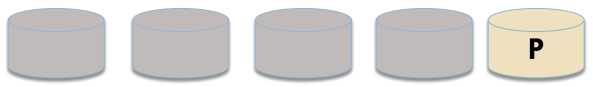
- Fewer I/Os per-second since every disk participates in every I/O
- Overheads for computing parity bits

RAID-4 Block interleaved parity

- Block-level striping
  - Block read accesses only one disk
  - Data transfer rate slower for each access
  - Multiple reads proceed in parallel
    - Higher overall I/O rate
- **Parity block** on a separate disk

49

RAID 4: Block interleaved parity



Parity block

If one disk fails

- Parity block used with corresponding blocks
  - Restore blocks of failed disk

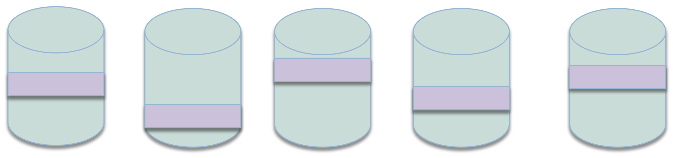
50

RAID-5
Block interleaved **distributed** parity

- **Spread** data and parity among all **`N+1`** disks
  - Avoid overuse of single parity disk
- Parity block does not store parity for blocks on the same disk

51

RAID 5:
Block interleaved distributed parity

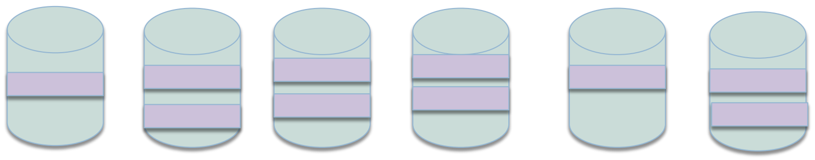


52

RAID-6

- Store extra redundant information
  - Guard against **multiple** disk failures
- Error correcting codes are used
  - Reed-Solomon codes
- 2-bits of redundant data
  - For every 4-bits of data

RAID-6



In the computer science department

- RAID 1
  - To mirror the root disks of the servers
- RAID 5
  - For all the "no_back_up" partitions
- RAID 6
  - For all data disks

The contents of this slide-set are based on the following references

- *Thomas Anderson and Michael Dahlin. Operating Systems: Principles & Practice. 2nd edition. ISBN: 978-0-9856735-2-9 [Chapter 12, 13]*
- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 10, 11]
- *Andrew S Tanenbaum and Herbet Bos. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 5]*
- *Chris Woodford. Flash Memory. http://www.explainthatstuff.com/flashmemory.html*