# Error detection and correction

- A layered approach
  - At the hardware level, checksums and device-level checks
    - remedy through error correcting codes
  - At the system level, redundancy, as in RAID
  - End-to-end checks at the file system level

# Storage device failures and mitigation - I

- Sector/page failure (i.e., Partial failure)
  - Data lost, rest of device operates correctly
    - Permanent (e.g. due to scratches) or transient (e.g., due to "high fly writes" producing weak magnetic fields, or write/read disturb errors)
    - Non recoverable read errors: in 2011, one bad sector/page per $10^{14}$ to $10^{18}$ bits read
  - Mitigations
    - data encoded with additional redundancy (error correcting codes + error notification)
    - for non recoverable read errors, remapping (device includes spare sectors/pages)
  - Pitfalls
    - non-recoverable error rates are negligible - 10% when reading a 2TB disk with a bad sector/$10^{14}$ bits
    - non-recoverable error rates are constant - they depend on load, age, workload
    - failures are independent - errors often correlated in time or space
    - error rates are uniform - different causes can contribute differently to nonrecoverable read errors

# Example: unrecoverable read errors

- Your 500GB laptop disk just crashed BUT you have just made a full backup on a 500GB disk
  - non recoverable read error rate: 1 sector/$10^{14}$ bits read
- What is the probability of reading successfully the entire disk during restore?

Expected number of failures while reading the data:

$$500\ \text{GB} \times \frac{8 \times 10^9\ \text{bits}}{\text{GB}} \times \frac{1\ \text{error}}{10^{14}\ \text{bits}} = 0.04$$

Alternatively...

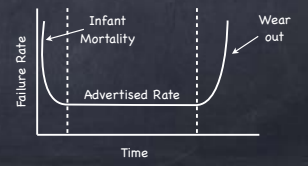
Assume each bit has a $10^{-14}$ chance of being wrong and that failures are independent

Probability to read all bits successfully:

$$(1 - 10^{-14})^{(500 \times 8 \times 10^9)} = 0.9608$$

# Storage device failures and mitigations - II

- Device failures
  - Device stops to be able to serve reads and writes to all sectors/pages (e.g. due to capacitor failure, damaged disk head, wear-out)
  - Annual failure rate
    - fraction of disks expected to fail/year
      - 2011: 0.5% to 0.9%
  - Mean Time To Failure (MTTF)
    - inverse of annual failure rate
      - 2011: $10^6$ hours (0.9%) to $1.7 \times 10^6$ hours (0.5%)
  - Pitfalls
    - MTTF measures a device's useful life (MTTF applies to device's intended service life)
    - advertised failure rates are trustworthy
    - failures are independent
    - failure rates are constant
    - devices behave identically
    - ignore warning signs (SMART technology)




Self Monitornig, Analysis, ReportTing

# Example: disk failures in a large system

- File server with 100 disks
- MTTF for each disk: $1.5 \times 10^6$ hours
- What is the expected time before one disk fails?

Assuming independent failures and constant failure rates:

MTTF for some disk = MTTF for single disk / 100 = $1.5 \times 10^4$ hours

Probability that some disk will fail in a year:

$$(365 \times 24) \text{ hours} \times \frac{1}{1.5 \times 10^4} \frac{\text{errors}}{\text{hours}} = 58.5\%$$

Pitfalls:

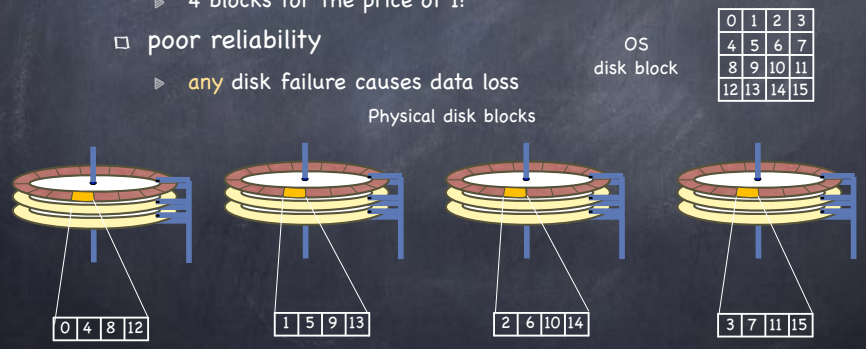
actual failure rate may be higher than advertised
failure rate may not be constant

# RAID

- Redundant Array of Inexpensive* Disks
  * In industry, "inexpensive" has been replaced by "independent" :-)
  - Disks are cheap, so put many (10s to 100s) of them in one box to increase storage, performance, and reliability
    - data plus some redundant information striped across disks
    - performance and reliability depend on how precisely it is striped
  - key feature: transparency
    - to the host system it all looks like a single, large, highly performant and highly reliable single disk
  - key issue: mapping
    - from logical block to location on one or more disks

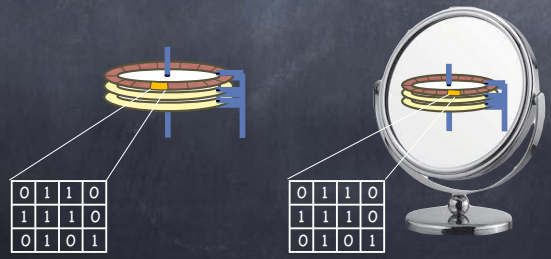
# RAID-0: High throughput, low reliability

- Disk striping (RAID-0)
  - higher disk bandwidth through larger effective block size
    - 4 blocks for the price of 1!
  - poor reliability
    - any disk failure causes data loss



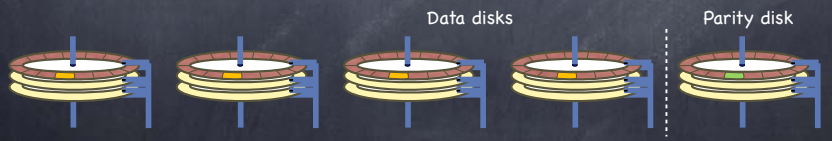

# RAID-1 mirrored disks

- Data written in two places
  - on failure, use surviving disk
- On read, choose fastest to read
- Expensive



# RAID-3

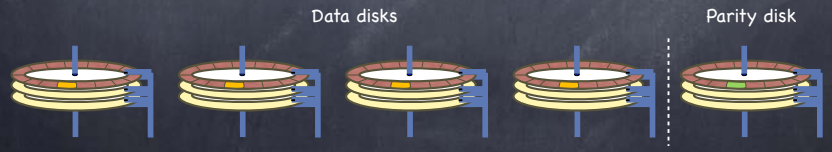
- Bit striped, with parity
  - given G disks,
    - parity = $data_0 \oplus data_1 \oplus \ldots \oplus data_{G-1}$
    - $data_0$ = parity $\oplus data_1 \oplus \ldots \oplus data_{G-1}$
- Reads access all data disks
- Writes accesses all data disks plus parity disk




Disk controller can identify faulty disk
- single parity disk can detect and correct errors

# RAID-4

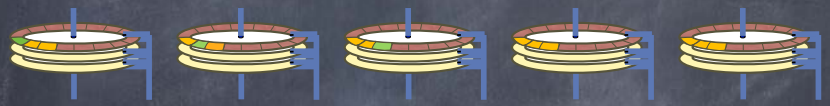
- Block striped, with parity
- Combines RAID-0 and RAID-3
  - reading a block accesses a single disk
  - writing always accesses parity disk
    - Heavy load on parity disk




Disk controller can identify faulty disk
- single parity disk can detect and correct errors

# RAID-5

- Block Interleaved Distributed Parity
  - no single disk dedicated to parity
  - parity and data distributed across all disks

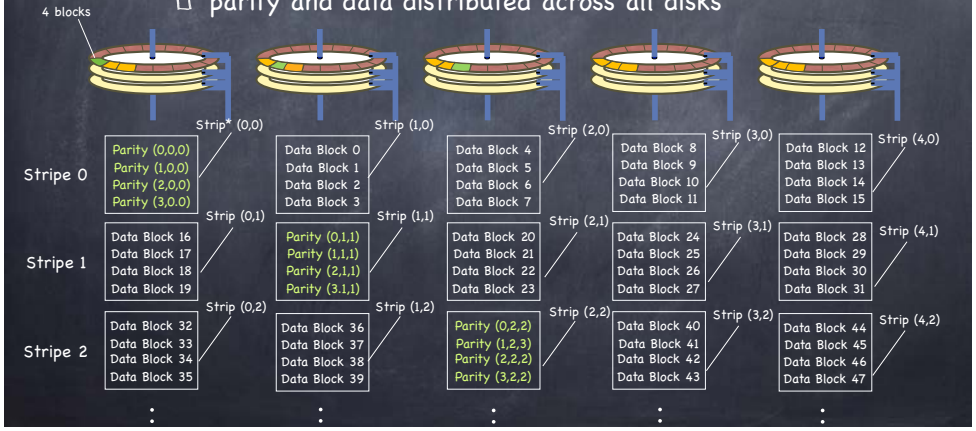


| | | | | |
|---|---|---|---|---|
| Parity 0-3 | Data 0 | Data 1 | Data 2 | Data 3 |
| Data 4 | Parity 4-7 | Data 5 | Data 6 | Data 7 |
| Data 8 | Data 9 | Parity 8-11 | Data 10 | Data 11 |
| Data 12 | Data 13 | Data 14 | Parity 12-15 | Data 15 |
| Data 16 | Data 17 | Data 18 | Data 19 | Parity 16-19 |
| : | : | : | : | : |

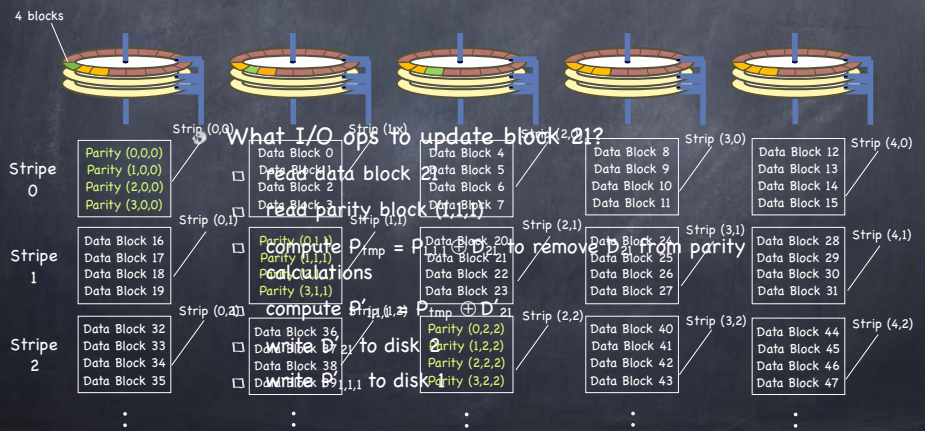
# RAID-5

- Block Interleaved Distributed Parity
  - no single disk dedicated to parity
  - parity and data distributed across all disks

Strip*: Sequence of sequential blocks that defines the unit of striping



# Example: Updating a RAID with rotating parity




- What I/O ops to update block 21?
  - read data block 21
  - read parity block (1,1,1)
  - compute $P_{tmp} = P_{1,1,1} \oplus D_{21}$ to remove $D_{21}$ from parity calculations
  - compute $P'_{1,1,1} = P_{tmp} \oplus D'_{21}$
  - write $D'_{21}$ to disk 2
  - write $P'_{1,1,1}$ to disk 1

# The File System abstraction

- Presents applications with persistent, named data
- Two main components:
  - files
  - directories

# The File

- A file is a named collection of data.
- A file has two parts
  - data – what a user or application puts in it
    - array of untyped bytes (in MacOS HFS, multiple streams per file)
  - metadata – information added and managed by the OS
    - size, owner, security info, modification time

# The Directory

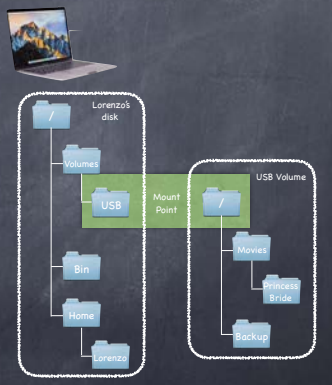
- The directory provides names for files
  - a list of human readable names
  - a mapping from each name to a specific underlying file or directory (hard link)
  - a soft link is instead a mapping from a file name to another file name
    - alias: a soft link that continues to remain valid when the (path of) the target file name changes

# Path and Volume

- Path: string that identifies a file or directory
  - absolute (if it stats with "/", the root directory)
  - relative (w.r.t. the current working directory)
- Volume: a collection of physical storage resources forming a logical storage device

# Mount

- Mount: allows multiple file systems on multiple volumes to form a single logical hierarchy
  - a mapping from some path in existing file system to the root directory of the mounted file system




# File system API

- Creating and deleting files
  - create() creates 1) a new file with some metadata and 2) a name for the file in a directory
  - link() creates a hard link–a new name for the same underlying file
  - unlink() removes a name for a file from its directory. If last link, file itself and resources it held are deleted
- Open and close
  - open() provides caller with a file descriptor to refer to file
    - permissions checked at open() time (a capability!)
    - creates per-file data structure, referred to by file descriptor
      - file ID, R/W permission, pointer to process position in file
  - close() releases data structure
- File access
  - read(), write(), seek()
    - but can use mmap() to create a mapping between region of file and region of memory
  - fsync() does not return until data is written to persistent storage

# Block vs Sector

- OS may choose block size larger than a sector on disk.
  - each block consists of consecutive sectors (why?)
    - larger block size increases transfer efficiency (why?)
    - can be handy to have block size equal page size (why?)

# File systems: What's so hard?

- Just map file name & offset keys to block numbers on a device values !

# File systems: What's so hard?

- Just map file name & offset keys to block numbers on a device values !
- Not so fast!
  - Performance
    - spatial locality
  - Flexibility
    - must handle diverse workloads and diverse file sizes
  - Reliability
    - must handle OS crashes and HW malfunctions