Guest Lecture for 15-440

Disk Array Data Organizations and RAID

Plan for today

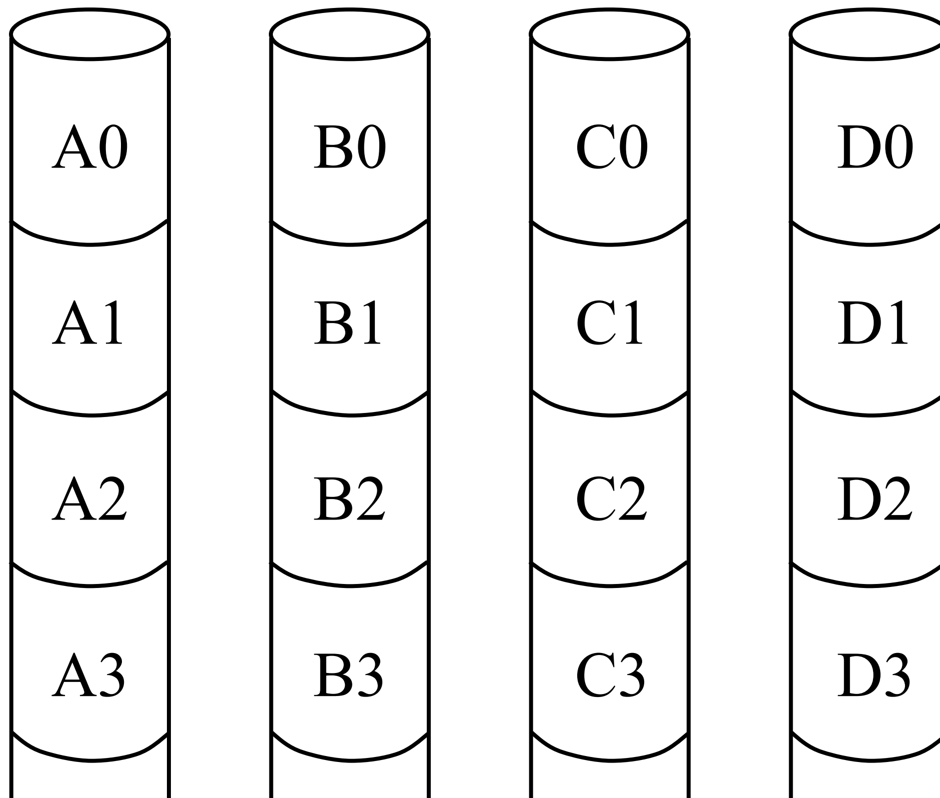
- Why have multiple disks?
  - Storage capacity, performance capacity, reliability
- Load distribution
  - problem and approaches
  - disk striping
- Fault tolerance
  - replication
  - parity-based protection
- “RAID” and the Disk Array Matrix
- Rebuild

Why multi-disk systems?

- A single storage device may not provide enough
  - storage capacity, performance capacity, reliability

So, what is the simplest arrangement?

Just a bunch of disks (JBOD)



- Yes, it's a goofy name
  - industry really does sell "JBOD enclosures"

Disk Subsystem Load Balancing

- I/O requests are almost never evenly distributed
  - Some data is requested more than other data
  - Depends on the apps, usage, time, …

Disk Subsystem Load Balancing

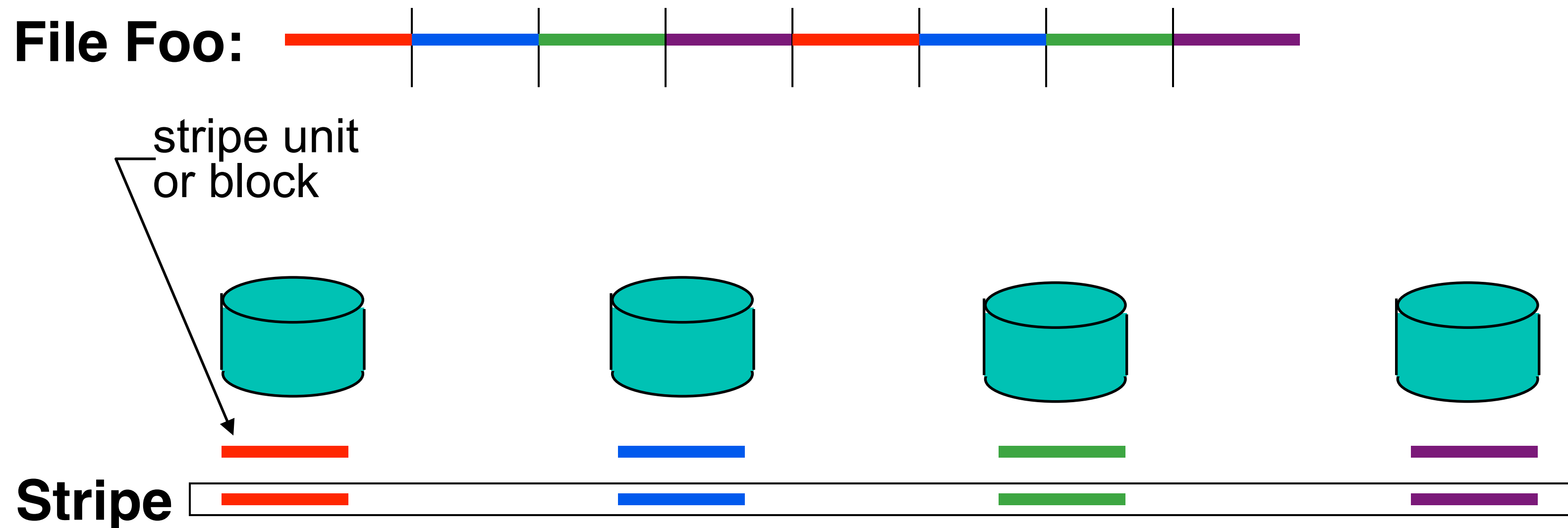
- I/O requests are almost never evenly distributed
  - Some data is requested more than other data
  - Depends on the apps, usage, time, …
- What is the right data-to-disk assignment policy?
  - Common approach: Fixed data placement
    - Your data is on disk X, period!
    - For good reasons too: you bought it or you're paying more …
  - Fancy: Dynamic data placement
    - If some of your files are accessed a lot, the admin (or even system) may separate the "hot" files across multiple disks
      - In this scenario, entire files systems (or even files) are manually moved by the system admin to specific disks

Disk Subsystem Load Balancing

- I/O requests are almost never evenly distributed
  - Some data is requested more than other data
  - Depends on the apps, usage, time, …
- What is the right data-to-disk assignment policy?
  - Common approach: Fixed data placement
    - Your data is on disk X, period!
  - Fancy: Dynamic data placement
    - If some of your files are accessed a lot, we may separate the "hot" files across multiple disks
      - In this scenario, entire files systems (or even files) are manually moved by the system admin to specific disks
  - Alternative: Disk striping
    - Stripe all of the data across all of the disks

Disk Striping

- Interleave data across multiple disks
  - Large file streaming can enjoy parallel transfers
  - High throughput requests can enjoy thorough load balancing
    - If blocks of hot files equally likely on all disks (really?)



Disk striping details

- How disk striping works
  - Break up total space into fixed-size stripe units
  - Distribute the stripe units among disks in round-robin
  - Compute location of block #B as follows
    - disk# = B % N (%=modulo, N = # of disks)
    - LBN# = B / N (computes the LBN on given disk)

Now, What If A Disk Fails?

- In a JBOD (independent disk) system
  - one or more file systems lost
- In a striped system
  - a part of each file system lost

- Backups can help, but
  - backing up takes time and effort (later in term)
  - backup doesn't help recover data lost during that day
    - any data loss is a big deal to a bank or stock exchange

And they do fail

Disk drive failure is top bugbear for IT pros

By: John Leyden
Posted: 13/03/2001 at 16:17 GMT

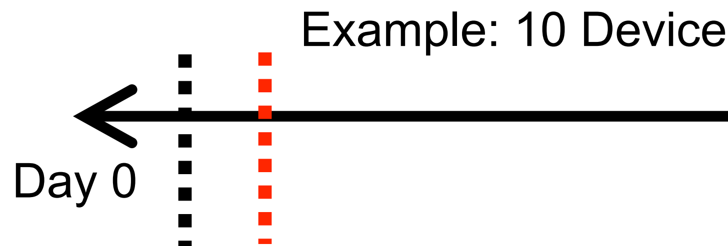
Hard drive crashes are the number one concern for systems administrators in charge of keeping storage systems up and running.

That's the conclusion of a survey of 900 IT professionals, 61 per cent of which rated hard-disk failure as their most pressing concern when it came to hard drive problems. Running out of disk drive space was cited as the second most important issue, and was rated as their top bug bear by 27 per cent of respondents to the survey.

The study, conducted by Survey.com for Executive Software, also reported that managers estimated their direct cost of a hard-drive failure at $15,000 per incident, a figure which doesn't include lost productivity and or the effects on sales while systems are down.

Sidebar: Reliability metric

- Mean Time Between Failures (MTBF)
  - Usually computed by dividing a length of time by the number of failures during that time (averaged over a large population of items)



Basically, we divide the time by the number of failures per device. This gives us the average time per failure per device.

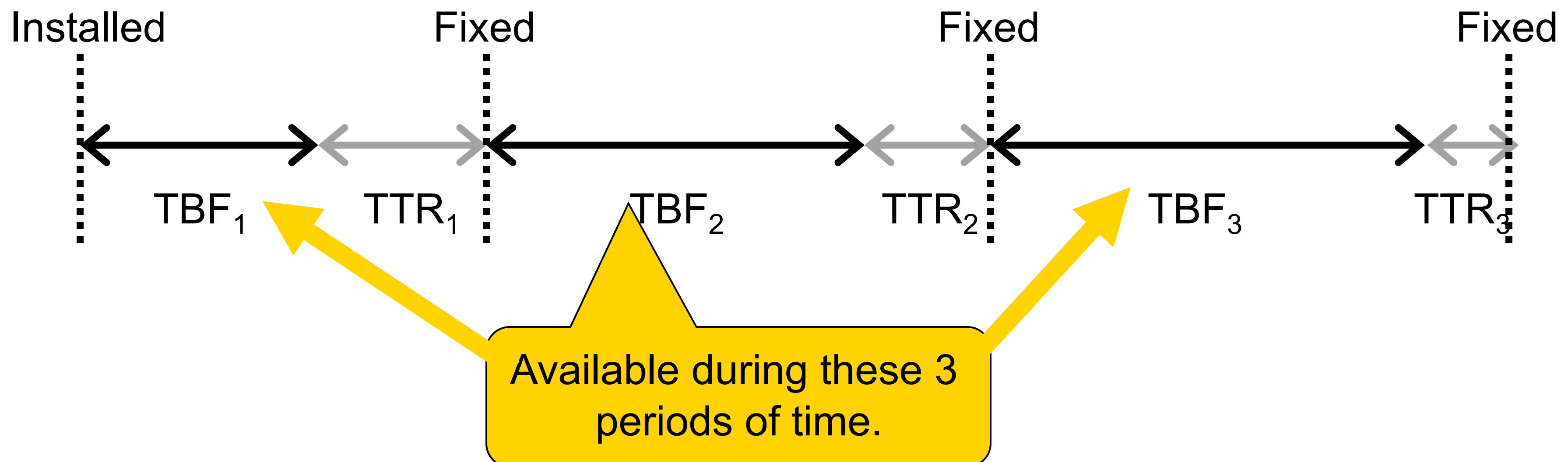
$$\text{MTBF} = \frac{\text{1000 Days}}{\text{5 failures / 10 devices}} = \text{2000 Days [per device]}$$

- Note: NOT a guaranteed lifetime for a particular item!

Sidebar: Availability metric

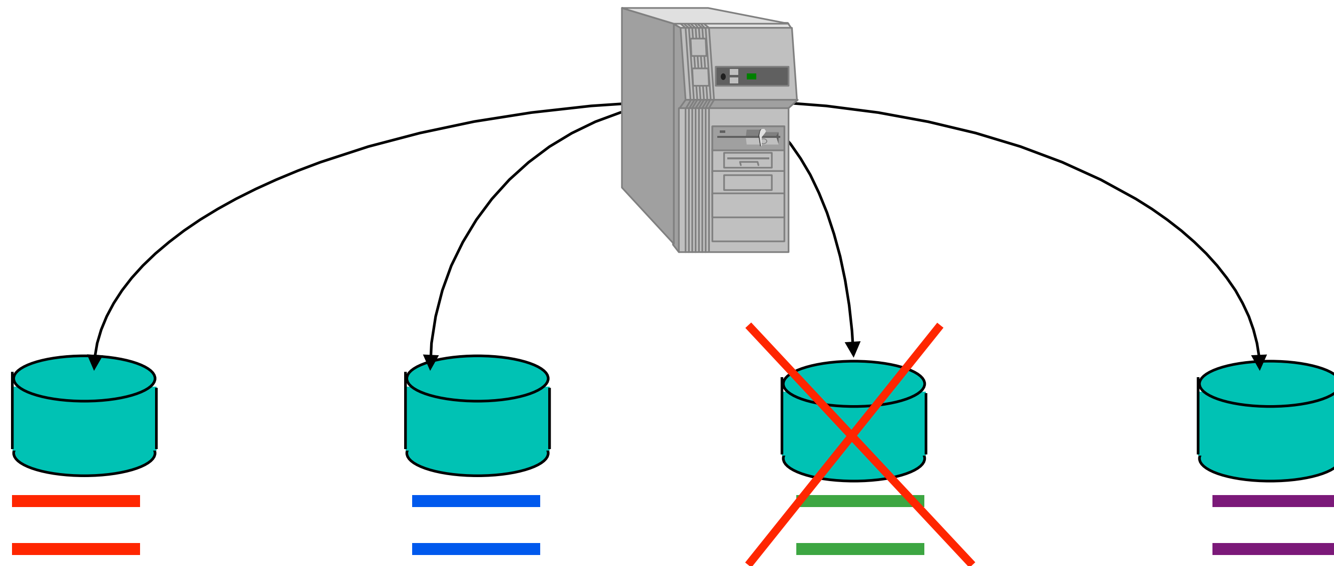
- Fraction of time that server is able to handle requests
  - Computed from MTBF and MTTR (Mean Time To Repair)

$$\textbf{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$



How often are failures?

- MTBF (Mean Time Between Failures)
  - $MTBF_{disk}$ ~ 1,200,00 hours (~136 years, <1% per year)
    - pretty darned good, if you believe the number
- $MTBF_{mutli\text{-}disk\ system}$ = mean time to first disk failure
  - which is $MTBF_{disk}$ / (number of disks)
  - For a striped array of 200 drives
    - $MTBF_{array}$ = 136 years / 200 drives = 0.65 years

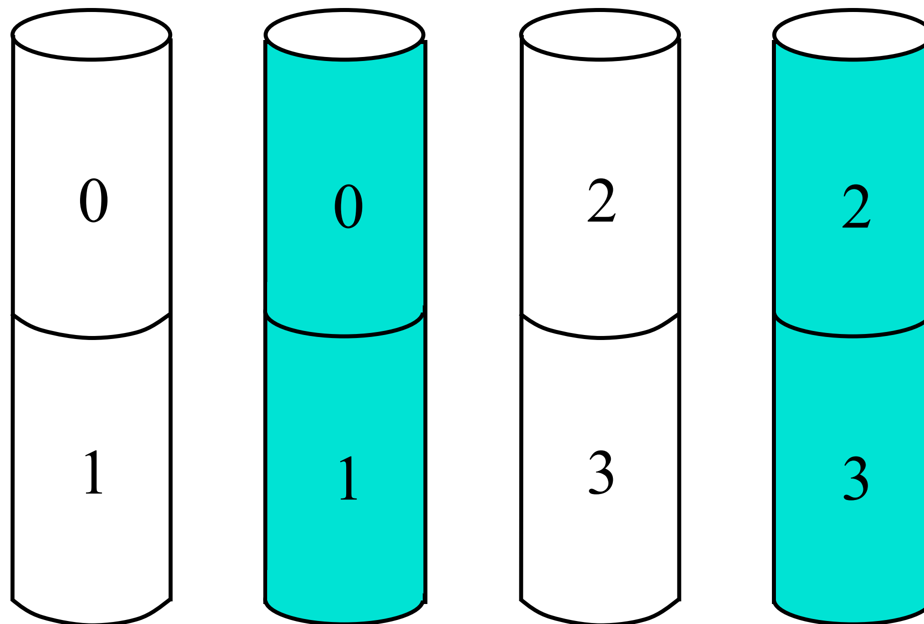


Tolerating and masking disk failures

- If a disk fails, it's data is gone
  - may be recoverable, but may not be
- To keep operating in face of failure
  - must have some kind of data redundancy
- Common forms of data redundancy
  - replication
  - erasure-correcting codes
  - error-correcting codes

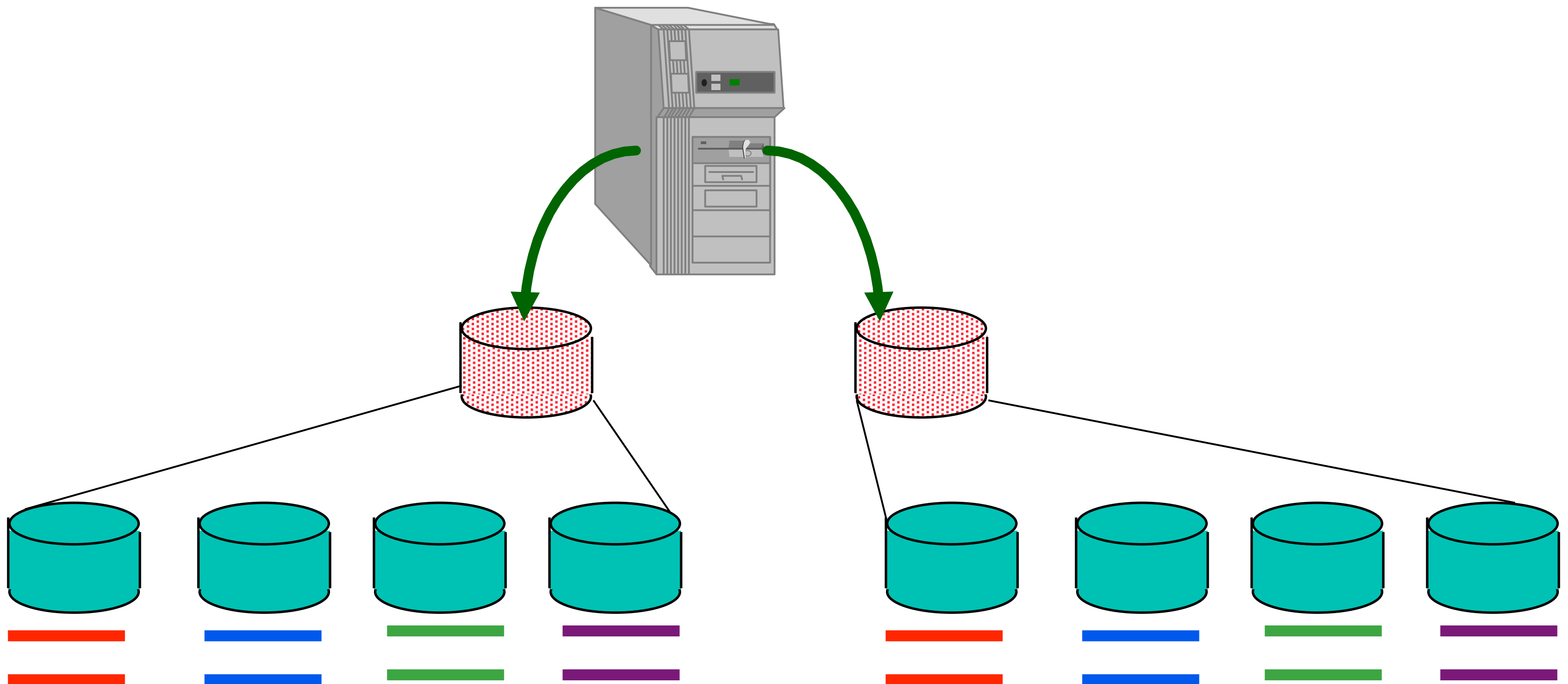
Redundancy via replicas

- Two (or more) copies
  - mirroring, shadowing, duplexing, etc.
- Write both, read either



Mirroring & Striping

- Mirror to 2 virtual drives, where each virtual drive is really a set of striped drives
  - Provides reliability of mirroring
  - Provides striping for performance (with write update costs)



Implementing Disk Mirroring

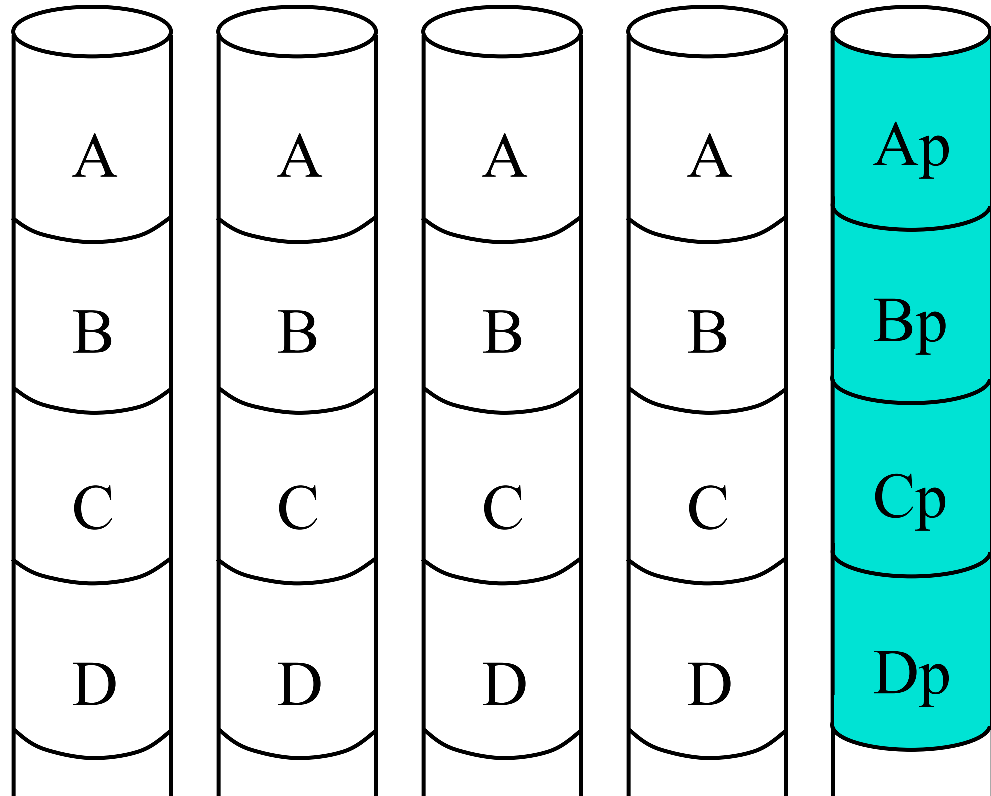
- Mirroring can be done in either software or hardware
- Software solutions are available in most OS's
  - Windows2000, Linux, Solaris
- Hardware solutions
  - Could be done in Host Bus Adaptor(s)
  - Could be done in Disk Array Controller

Lower Cost Data Redundancy

- Single failure protecting codes
  - general single-error-correcting code is overkill
    - General code finds error and fixes it
  - disk failures are self-identifying (a.k.a. erasures)
    - Don't have to find the error
  - fact: N-error-detecting code is also N-erasure-correcting
    - Error-detecting codes can't find an error, just know its there
    - But if you independently know where error is, allows repair
- Parity is single-disk-failure-correcting code
  - recall that parity is computed via XOR
  - it's like the low bit of the sum

Simplest approach: Parity Disk

- One extra disk
- All writes update parity disk
  - potential bottleneck



Aside: What's Parity?

- Parity
  - count number of 1's in a byte and store a parity bit with each byte of data
    - Solve equation XOR-sum(data bits, parity) = 0
  - parity bit is computed as
    - If the number of 1's is odd, store a 1
    - If the number of 1's is even, store a 0
    - This is called even parity (# of ones is even)
  - Example:

    0x54 == $0101\ 0100_2$ (Three 1's --> parity bit is set to "1")

    Store 9 bits: 0101 0100 1

  - Enables:
    - Detection of single-bit errors (equation won't work if one bit flipped)
    - Reconstruction of single-bit erasures (reverse equation for unknown)

Aside: What's Parity (con't)

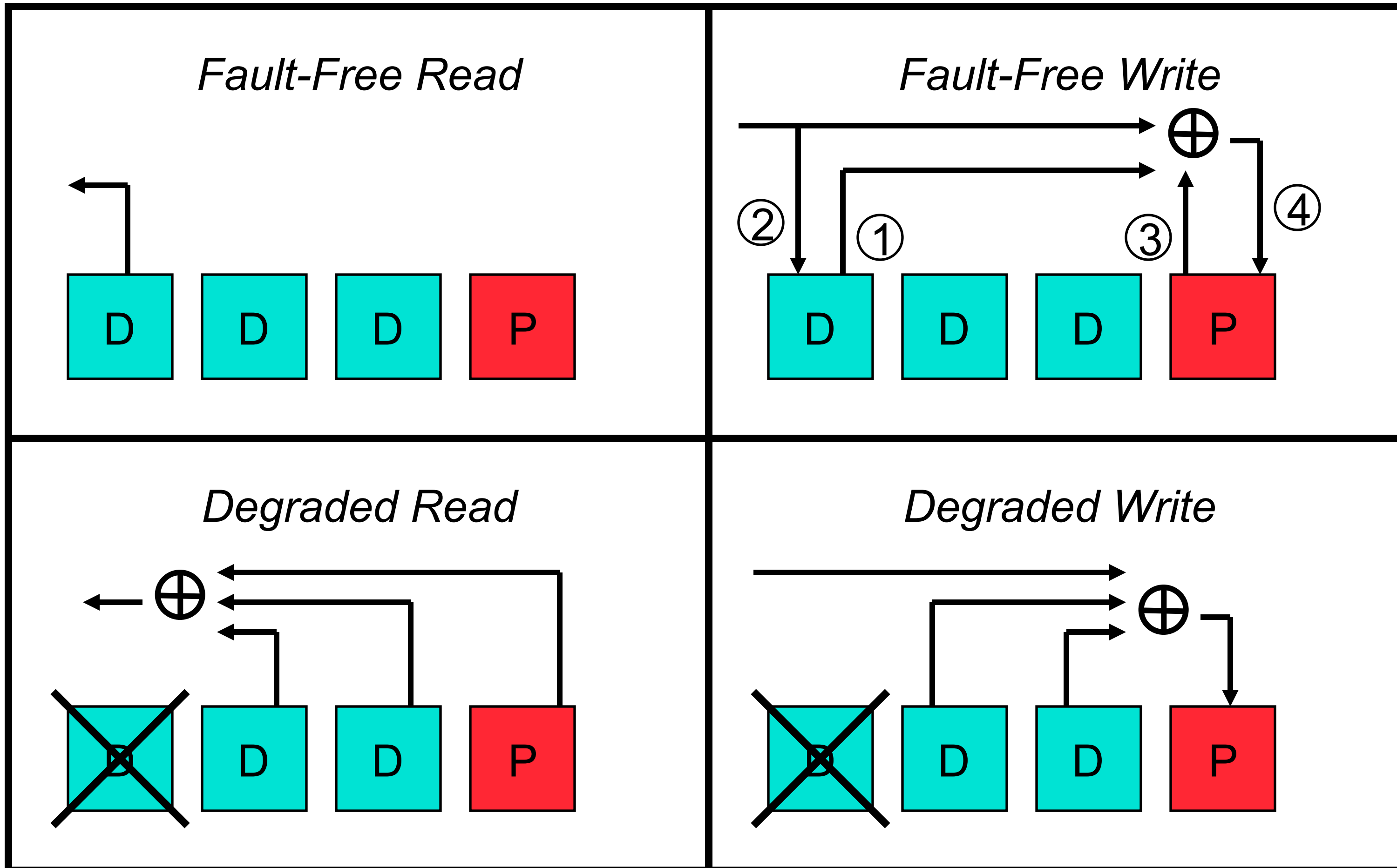
- Example

  0x54 == 0101 0100$_2$ (Three 1's --> parity bit is set to "1")
  Store 9 bits: 0101 0100 1

  - What if we want to update bit 3 from "1" to "0"
    - Could completely recompute the parity
    - 0x54 → 0x44

      0x44 == 0100 0100$_2$ (Two 1's --> parity bit is set to "0")
      Store 9 bits: 0100 0100 0
    - Or, we could subtract out old data, then add in new data
      - How do we subtract out old data?
        - oldData XOR oldParity
        - 1 XOR 1 == 0 ← this is parity w/out dataBit3
      - How do we add in new data
        - newData XOR newParity
        - 0 XOR 0 == 0 ← this is parity w/new dataBit3
  - Therefore, updating new data doesn't require one to re-read all of the data
    - Or, for each data bit that "toggles", toggle the parity bit

Updating and using the parity

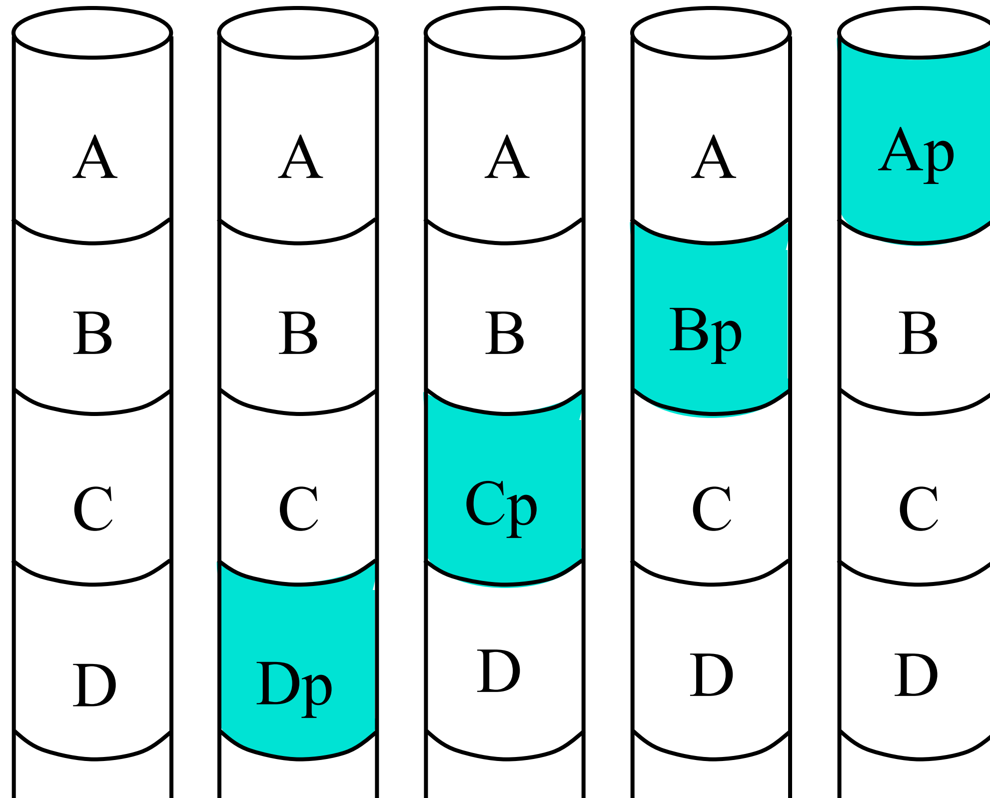


The parity disk bottleneck

- Reads go only to the data disks
  - But, hopefully load balanced across the disks
- All writes go to the parity disk
  - And, worse, usually result in Read-Modify-Write sequence
  - So, parity disk can easily be a bottleneck

Solution: Striping the Parity

- Removes parity disk bottleneck



RAID Taxonomy

- Redundant Array of Inexpensive Independent Disks
  - Constructed by UC-Berkeley researchers in late 80s (Garth)
- RAID 0 – Course-grained Striping with no redundancy
- RAID 1 – Mirroring of independent disks
- RAID 2 – Fine-grained data striping plus Hamming code disks
  - Uses Hamming codes to detect and correct multiple errors
  - Originally implemented when drives didn't always detect errors
  - Not used in real systems
- RAID 3 – Fine-grained data striping plus parity disk
- RAID 4 – Course-grained data striping plus parity disk
- RAID 5 – Course-grained data striping plus striped parity
- RAID 6 – Course-grained data striping plus 2 striped codes

RAID 6

- P+Q Redundancy
  - Protects against multiple failures using Reed-Solomon codes
  - Uses 2 “parity” disks
    - P is parity
    - Q is a second code
    - It’s two equations with two unknowns, just make “bigger bits”
      - Group bits into “nibbles” and add different co-efficients to each equation (two independent equations in two unknowns)
  - Similar to parity striping
    - De-clusters both sets of parity across all drives
    - For small writes, requires 6 I/Os
      - Read old data, old parity1, old parity2
      - Write new data, new parity1, new parity2

Disk array subsystems

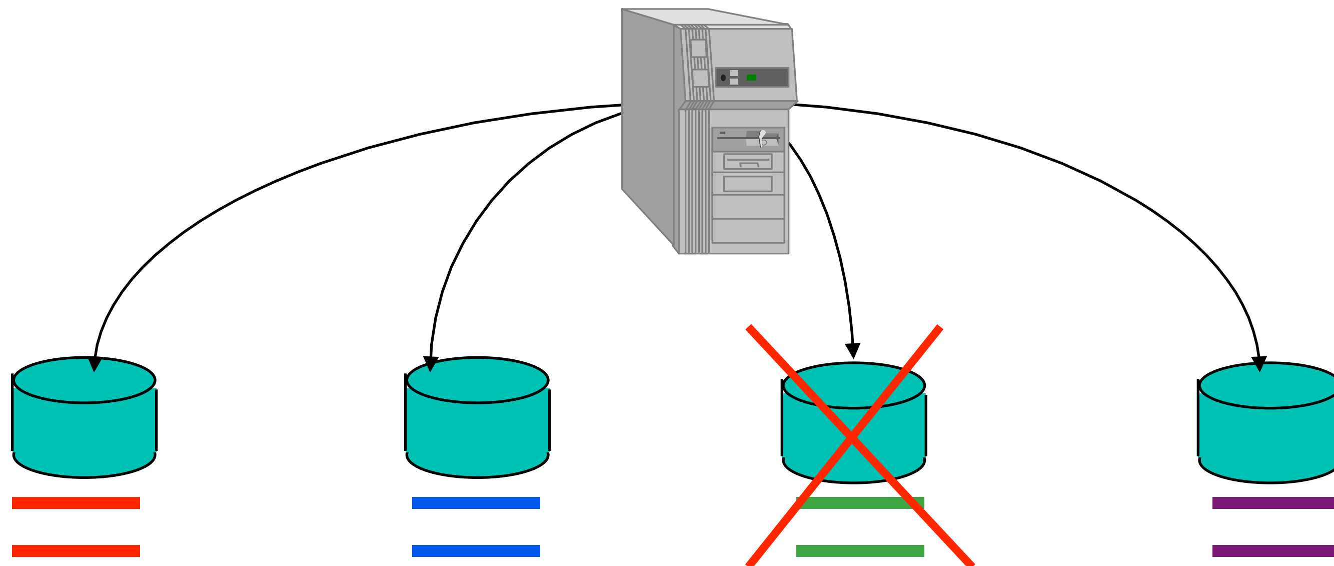
- Sets of disks managed by a central authority
  - e.g., file system (within OS) or disk array controller
- Data distribution
  - squeezing maximum performance from the set of disks
  - several simultaneous considerations
    - intra-access parallelism: parallel transfer for large requests
    - inter-access parallelism: concurrent accesses for small requests
    - load balancing for heavy workloads
- Redundancy scheme
  - achieving fault tolerance from the set of disks
  - several simultaneous considerations
    - space efficiency
    - number/type of faults tolerated
    - performance

The Disk Array Matrix

	Independent	Fine Striping	Course Striping
None	JBOD		RAID0
Replication	Mirroring RAID1		RAID0+1
Parity Disk		RAID3	RAID4
Striped Parity	Gray90		RAID5

Back to Mean Time To Data Loss (MTTDL)

- MTBF (Mean Time Between Failures)
  - $MTBF_{disk}$ ~ 1,200,00 hours (~136 years)
    - pretty darned good, if you believe the number
- $MTBF_{mutli\text{-}disk\ system}$ = mean time to first disk failure
  - which is $MTBF_{disk}$ / (number of disks)
  - For a striped array of 200 drives
    - $MTBF_{array}$ = 136 years / 200 drives = 0.65 years



Reliability without rebuild

- 200 data drives with $MTBF_{drive}$
  - $MTTDL_{array} = MTBF_{drive} / 200$
- Add 200 drives and do mirroring
  - $MTBF_{pair} = (MTBF_{drive} / 2) + MTBF_{drive} = 1.5 * MTBF_{drive}$
  - $MTTDL_{array} = MTBF_{pair} / 200 = MTBF_{drive} / 133$
- Add 50 drives, each with parity across 4 data disks
  - $MTBF_{set} = (MTBF_{drive} / 5) + (MTBF_{drive} / 4) = 0.45 * MTBF_{drive}$
  - $MTTDL_{array} = MTBF_{set} / 50 = MTBF_{drive} / 111$

Rebuild: restoring redundancy after failure

- After a drive failure
  - data is still available for access
  - but, a second failure is BAD
- So, should reconstruct the data onto a new drive
  - on-line spares are common features of high-end disk arrays
    - reduce time to start rebuild
  - must balance rebuild rate with foreground performance impact
    - a performance vs. reliability trade-offs
- How data is reconstructed
  - Mirroring: just read good copy
  - Parity: read all remaining drives (including parity) and compute

Reliability consequences of adding rebuild

- No data loss, if fast enough
  - That is, if first failure fixed before second one happens
- New math is…
  - $MTTDL_{array} = MTBF_{firstdrive}$ * (1 / prob of 2nd failure before repair)
  - … which is $MTTR_{drive} / MTBF_{seconddrive}$
- For mirroring
  - $MTBF_{pair} = (MTBF_{drive} / 2) * (MTBF_{drive} / MTTR_{drive})$
- For 5-disk parity-protected arrays
  - $MTBF_{set} = (MTBF_{drive} / 5) * (MTBF_{drive} / 4 / MTTR_{drive})$

Three modes of operation

- Normal mode
  - everything working; maximum efficiency
- Degraded mode
  - some disk unavailable
  - must use degraded mode operations

Three modes of operation

- Normal mode
  - everything working; maximum efficiency
- Degraded mode
  - some disk unavailable
  - must use degraded mode operations
- Rebuild mode
  - reconstructing lost disk's contents onto spare
  - degraded mode operations plus competition with rebuild

Mechanics of rebuild

- Background process
  - use degraded mode read to reconstruct data
  - then, write it to replacement disk

- Implementation issues
  - Interference with foreground activity and controlling rate
    - rebuild is important for reliability
    - foreground activity is important for performance
  - Using the rebuilt disk
    - for rebuilt part, reads can use replacement disk
    - must balance performance benefit with rebuild interference