15-410

"...Failure is not an option..."

Disk Arrays
Mar. 16, 2011

Garth Gibson

Dave Eckhardt

Contributions by
Michael Ashley-Rollman

Synchronization

Today: Disk Arrays

- **Text: 12.7 (a good start)**
  - **Please read remainder of chapter too**
- **Papers (@end)**

Friday

- **Checkpoint 2, in cluster, as last time**
- **Attendance is compulsory regardless of completion!**

Overview

Historical practices

- **Striping**
- **Mirroring**

The reliability problem

Parity, ECC, why parity is enough

RAID “levels” (really: flavors)

Applications

Papers

Striping

Goal

- High-performance I/O for databases, supercomputers
- “People with more money than time”

Problems with disks

- Seek time
- Rotational delay
- Transfer time

Seek Time

Technology issues evolve slowly

- **Weight of disk head**
- **Stiffness of disk arm**
- **Positioning technology**

Hard to dramatically improve for niche customers

Sorry!

Rotational Delay

How fast *can* we spin a disk?

- **Fancy motors, lots of power – spend more money**

Probably limited by data rate

- **Spin faster ⇒ must process analog waveforms faster**
- **Analog ⇒ digital via *serious* signal processing**

Special-purpose disks generally spin *a little* faster

- **1.5X, 2X – not 100X**

Transfer Time

Transfer time ≡

- **Assume seek & rotation complete**
- **How fast to transfer ____ kilobytes?**

We struck out on seek, rotation

- **Can we at least *transfer* faster than commodity disks?**

Parallel Transfer?

Reduce transfer time (without spinning faster)

Read from multiple heads at same time?

Practical problem

- **Disk needs N copies of analog ⇒ digital hardware**
- **Expensive, but we have *some* money to burn**

Marketing wants to know...

- **Do we have *enough* money to buy a new factory?**
- **Can't we use our existing product somehow?**

Striping

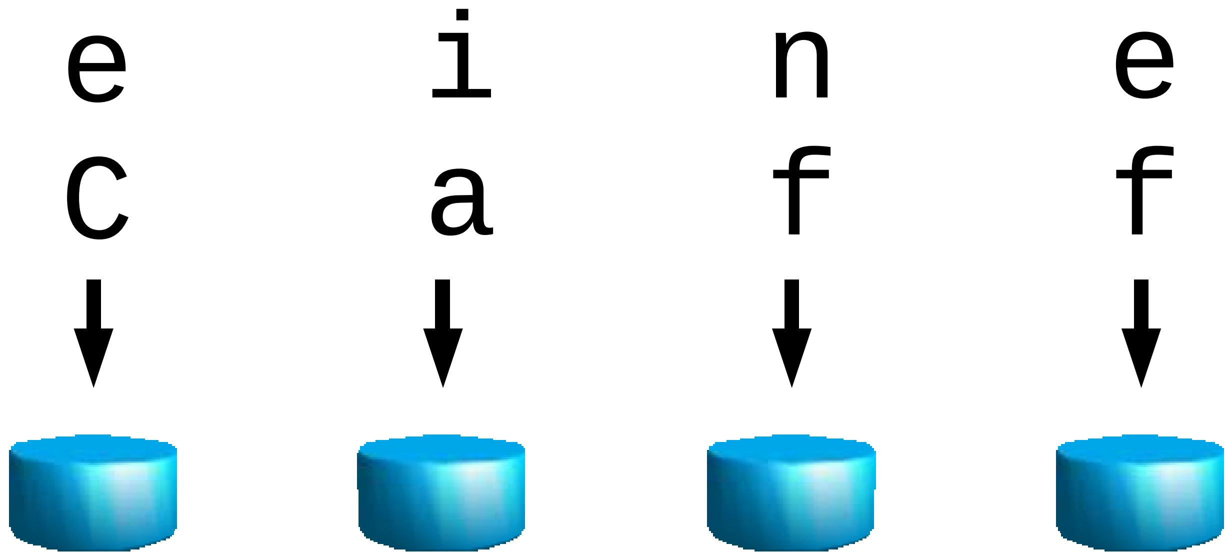
Goal

- **High-performance I/O for databases, supercomputers**

Solution: parallelism

- **Gang *multiple disks* together**

Striping



Striping

Stripe *unit* (what each disk gets) can vary

- **Byte**
- **Bit**
- **Sector**
- **“Block” of sectors (typically 4-64KB, 64MB in cloudFS)**

Stripe *size* = (stripe unit) X (#disks)

Behavior: “fat sectors”

- **File system maps bulk data request ⇒ N disk operations**
- **Each disk reads/writes 1 sector**

Striping Example

Simple case – stripe sectors

- 4 disks, stripe unit = 512 bytes
- Stripe size = 2K

Results

- Seek time: 1X base case (ok)
- Transfer rate: 4X base case (great!)

But there's a problem...

High-Performance Striping

Rotational delay *gets worse*

- **Stripe not done until** *fourth* **disk rotates to right place**
- **I/O to 1 disk pays** *average* **rotational delay (50%)**
- **N disks converge on** *worst-case* **rotational delay (100%)**

High-Performance Striping

Rotational delay *gets worse*

- Stripe not done until *fourth* disk rotates to right place
- I/O to 1 disk pays *average* rotational delay (50%)
- N disks converge on *worst-case* rotational delay (100%)

Spindle synchronization!

- Make sure N disks are always aligned
- All sector 0's pass under their heads at the “same” time

Result

- Commodity disks with extra synchronization hardware
  - Not *insanely* expensive ⇒ some supercomputer applications
  - Seagate ST15150W (4G, 1995), IBM UltraStar 2XP (9G, 1997)

Less Esoteric Goal: Capacity

Users always want more disk space

Easy answer

- **Build a larger disk!**
- **IBM 3380 (early 1980's)**
  - **14-inch platter(s)**
  - **Size of a *washing machine***

Less Esoteric Goal: Capacity

Users always want more disk space

Easy answer

- **Build a larger disk!**
- **IBM 3380 (early 1980's)**
  - **14-inch platter(s)**
  - **Size of a *washing machine***
  - **1-3 GByte (woo!)**

Less Esoteric Goal: Capacity

Users always want more disk space

Easy answer

- Build a larger disk!
- IBM 3380 (early 1980's)
  - 14-inch platter(s)
  - Size of a *washing machine*
  - 1-3 GByte (woo!)

"Marketing on line 1"...

- These monster disks sure are expensive to build!
  - Especially compared to those dinky 5¼-inch PC disks...
- Can't we hook small disks together like we did for speed?

Striping Example Revisited

Simple case – stripe sectors

- 4 disks, stripe unit = 512 bytes
- Stripe size = 2K

Results

- Seek time: 1X base case (ok)
- Rotation time : 1X base case using special hardware (ok)
- Transfer rate: 4X base case (great!)
- Capacity: 4X base case (great!)

Now **what could go wrong?**

The Reliability Problem

MTTF = Mean time to failure

MTTF(array) = MTTF(disk) / #disks

Example from original 1988 RAID paper

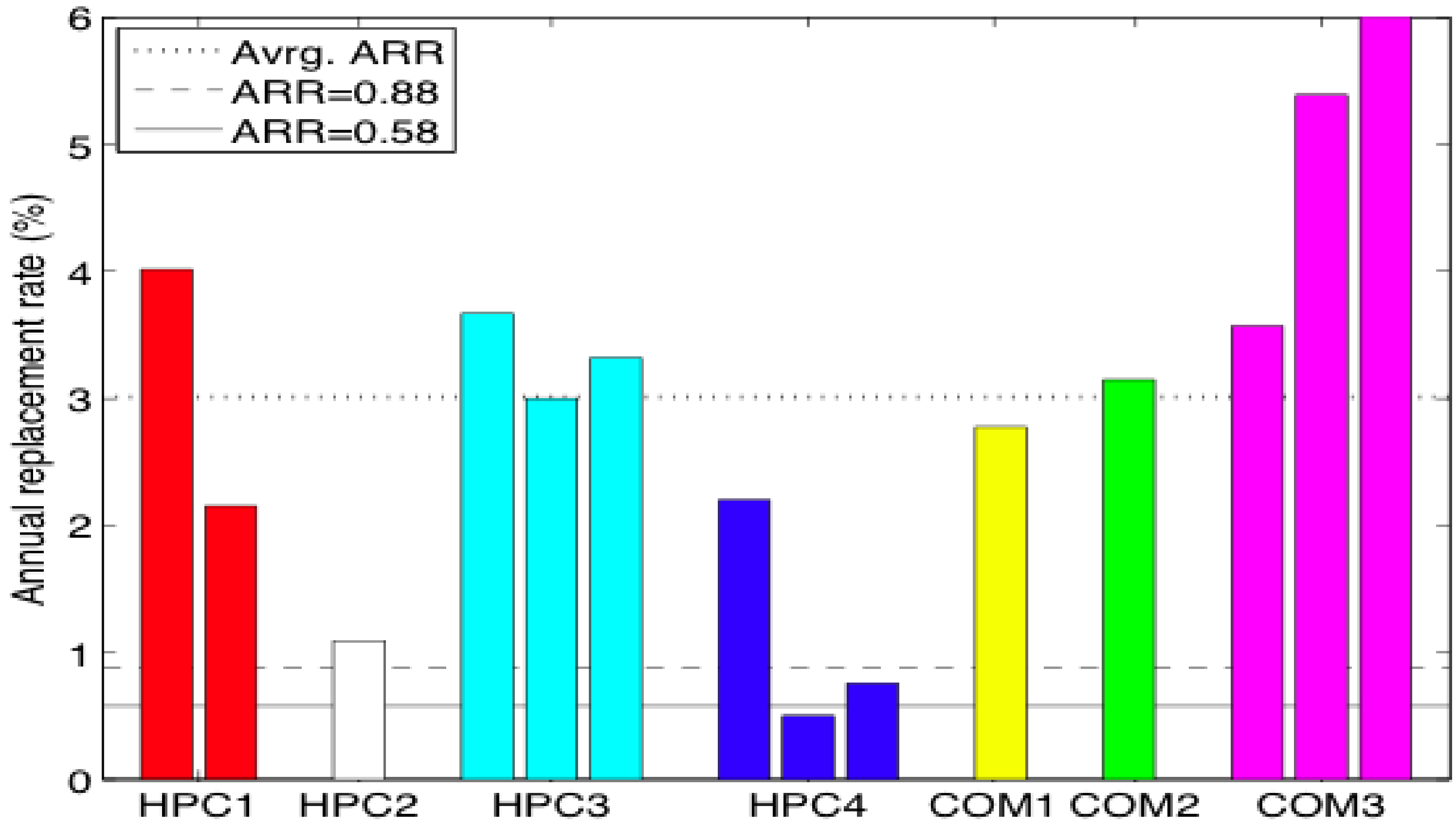
- **Conner Peripherals CP3100 (100 megabytes!)**
- **MTTF = 30,000 hours = 3.4 years**

Array of 100 CP3100's

- **10 Gigabytes (good)**
- **MTTF = 300 hours =** *12.5 days* **(not so good)**
- **Reload file system from tape every 2 weeks???**

Note: array MTTF is really more complicated than 1/N

Disk Failures



Schroeder & Gibson: “Disk failures in the real world”

Mirroring

We are computer scientists

- **Solve reliability via ...?**

Mirroring

We are computer scientists

- **Solve reliability via induction!**

Mirroring

We are computer scientists

- Solve reliability via induction!

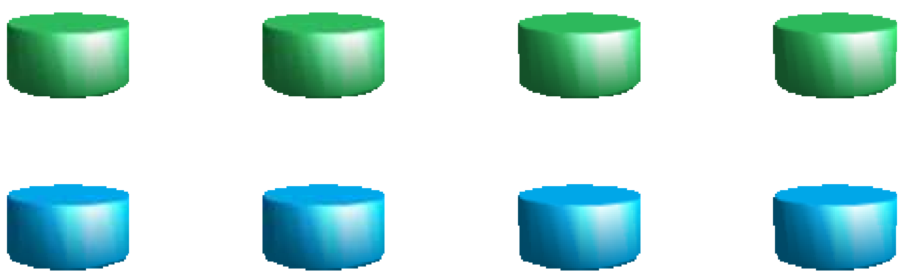
When a disk goes bad

- Base case: “Assume another disk contains the same bits”
- Induction: Copy bits from backup disk to a new blank disk

Restoring disks from tape is no fun

- Restoring disks from other disks is closer to fun

Mirroring



Copy A

Copy B

Mirroring

Operation

- **Write: write to *both* mirrors**
- **Read: read from *either* mirror**

Cost per byte *doubles*

Performance

- **Writes: a little slower**
- **Reads: maybe 2X faster**

Reliability *vastly* increased

Mirroring

When a disk breaks

- Identify it to system administrator
  - Beep, blink a light
- System administrator provides blank disk
- Copy contents from surviving mirror

Result

- Expensive but safe
- Banks, hospitals, etc.
- Home PC users???

Error Coding

If you are good at math

- Error Control Coding: Fundamentals & Applications
  - Lin, Shu, & Costello

If you are like Eckhardt

- Commonsense Approach to the Theory of Error Correcting Codes
  - Arazi

Error Coding In One Easy Lesson

Data vs. message

- **Data = what you want to convey**
- **Message = data plus extra bits ("code word")**

Error detection

- **Message indicates: something got corrupted**

Error *correction*

- **Message indicates: bit 37 should be 0, not 1**
- **Very useful!**

Trivial Example

Transmit *code words* instead of data bits

- Data 0 ≡ code word 0000
- Data 1 ≡ code word 1111

Transmission "channel" corrupts code words

- Send 0000, receive 0001

Error detection

- 0001 isn't a valid code word - Error!

Error *correction*

- Gee, 0001 looks more like "0000" than "1111"

Lesson 1, Part B

Error codes can be overwhelmed

- Is "0011" a corrupted "0000" or a corrupted "1111"?
- We know something is wrong, but we don't know what

"Too many" errors: *wrong answers*

- Series of corruptions
  - 0000 $\Rightarrow$ 0001 $\Rightarrow$ 0101 $\Rightarrow$ 1101
  - "Looks like 1111, doesn't it?"

Codes typically detect more errors than can correct

- A possible example code
  - Can ***detect*** 1..4 errors, can ***fix*** any single error
  - Five errors will report false "fix" - to a *different* user data word!

Parity

Parity = XOR "sum" of bits

- **$0 \oplus 1 \oplus 1 = 0$**

Parity provides *single error detection*

- **Sender transmits** *code word* **including data and** *parity bit*
- **Correct: 011,0**
- **Incorrect: 011,1**
  - **Something is wrong with this picture –** *but what?*
  - **Parity provides *no* error correction**

***Cannot* detect (all) multiple-bit errors**

ECC

ECC = error correcting code

“Super parity”

- **Code word: user data plus *multiple* “parity” bits**
- **Mysterious math computes parity from data**
  - **Hamming code, Reed-Solomon code**
- **Can detect N *multiple-bit* errors**
- **Can *correct* M (< N) bit errors!**
- **Often M ~ N/2**

Parity revisited

Parity provides single *erasure* correction!

Erasure channel

- **"Knows when it doesn't know something"**
- **Example: each bit is 0 or 1 or "don't know"**
- **Sender provides (user data, parity bit): (0 1 1 , 0)**
- **Channel provides corrupted message: (0 ? 1 , 0)**
  - *?* = $0 \oplus 1 \oplus 0 = 1$

Are erasure channels real??

Erasure channel???

Radio

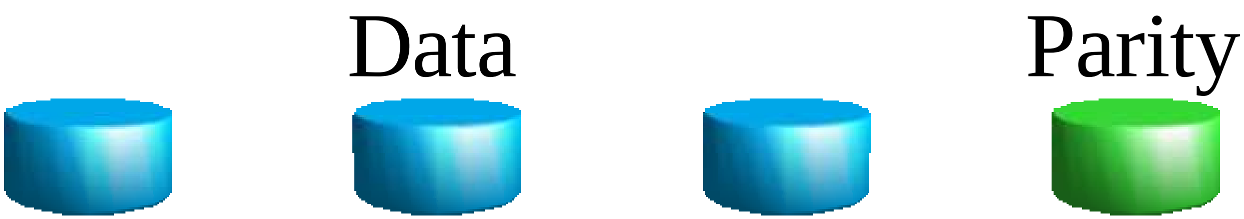
- **Modem stores signal strength during reception of each bit**

Erasure channel???

Disk drives!

- Disk hardware adds ECC data to each sector
  - Very good at detecting & correcting lots of bit corruption
  - When ECC can’t repair bit errors, a sector is lost
- Disks “know when they don't know”
  - Read sector 4271 from 4 different disks?
  - Receive N good sectors, 4-N errors (“sector erasures”)
- “Drive not ready”?
  - Maybe the drive's computer is broken...
  - Maybe motor/arm/head diagnostics have failed
  - No problem... consider every sector “erased”

"Fractional mirroring"

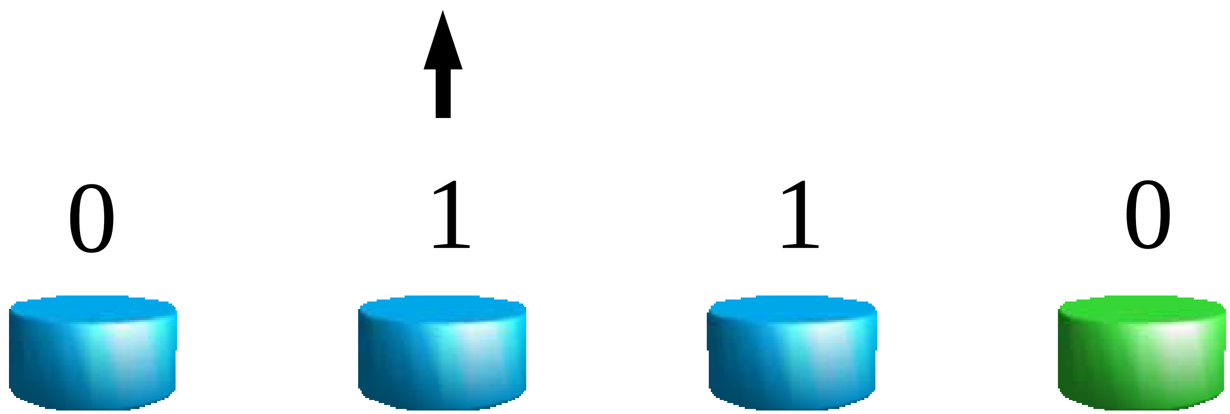


“Fractional mirroring”

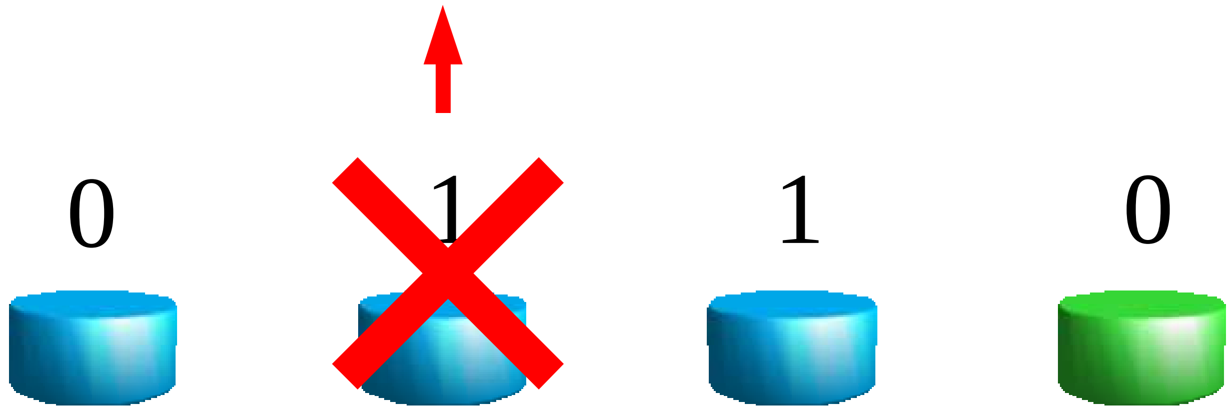
Operation

- **Read: read data disks**
  - **Error? Read parity disk, compute lost value**
- **Write: write data disks** *and parity disk*

Read



Read Error



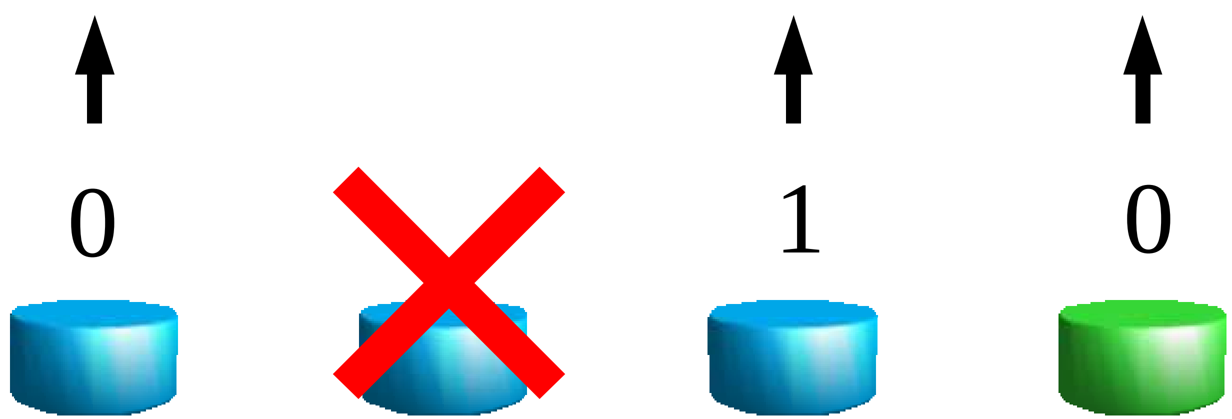
Read Reconstruction

Disk reports bit is missing

Read rest of bits in parity equation

Missing bit = XOR of surviving bits

- **Missing bit = $0 \oplus 1 \oplus 0 = 1$**



Missing $= 0 \oplus 1 \oplus 0 = 1$

“Fractional mirroring”

Performance

- **Reads: run at normal disk speed**
- **Writes: slower (see “RAID 4” below)**

Reliability *vastly* increased

- **Not quite as good as mirroring**
  - **Why not?**

Cost

- *Fractional* **increase (50%, 33%, ...)**
- **Cheaper than mirroring's 100%**

RAID

RAID

- Redundant Arrays of Inexpensive Disks
- Redundant Arrays of Independent Disks

SLED

- Single Large Expensive Disk

Terms from original RAID paper (@end)

Different ways to aggregate disks

- Paper presented a number-based taxonomy
- Metaphor stretched too far now

RAID “levels”

They're not really levels

- RAID 2 isn't “more advanced than” RAID 1
  - People really do RAID 1
  - People basically never do RAID 2

People invent new ones which don't sort well

- RAID 0+1 ???
- JBOD ???

Easy cases

JBOD = "just a bunch of disks"

- N disks in a box pretending to be 1 large disk
- Box controller maps "logical sector" ⇒ (disk, real sector)

Legacy approaches

- RAID 0 = striping
- RAID 1 = mirroring

RAID 2

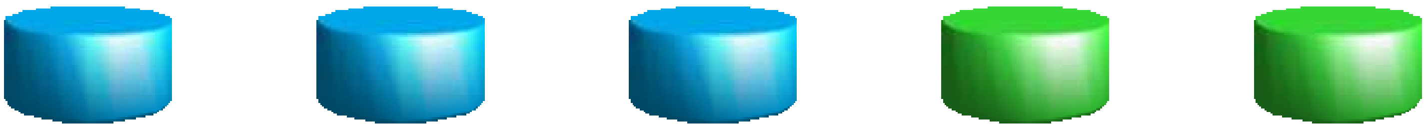
Stripe size = *"word"* (unit = 1 bit per disk)

N data disks, M parity disks

Use ECC to get multiple-error correction

Very rarely used

- **Thinking Machines SIMD hypercube machines in 1980's**



RAID 3

Stripe size = *"word"* (unit = 1 bit per disk)

Use parity instead of ECC (disks report erasures)

N data disks, 1 parity disk

- **Read from all N+1disks every time**

Used in some high-performance applications

- **Can do "on the fly repair" and "detect lying disks"**

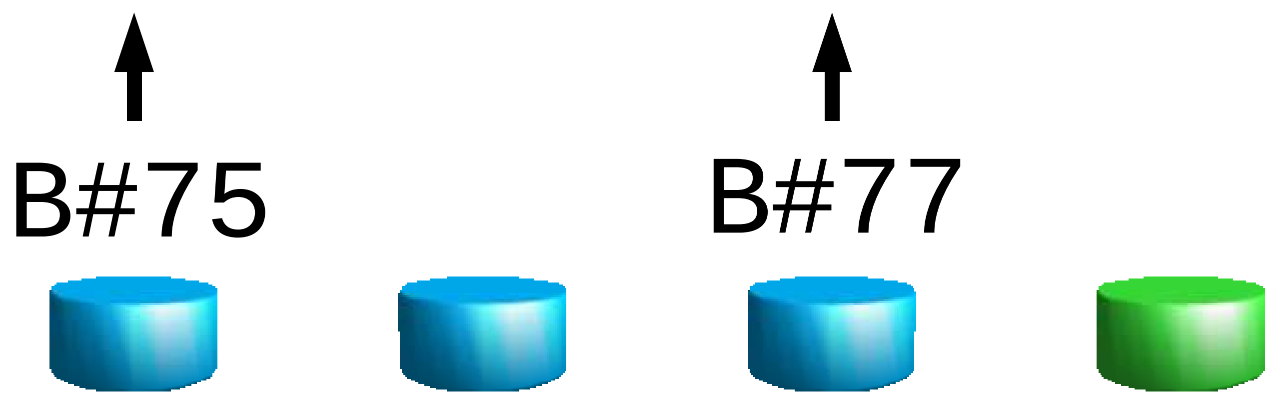
RAID 4

Like RAID 3

- Uses parity, relies on erasure signals from disks
- But unit = *block* instead of *bit*

Single-block reads involve only 1 disk!

- Can support single-block reads on different disks in parallel
  - Good for transaction processing, small files, high concurrency



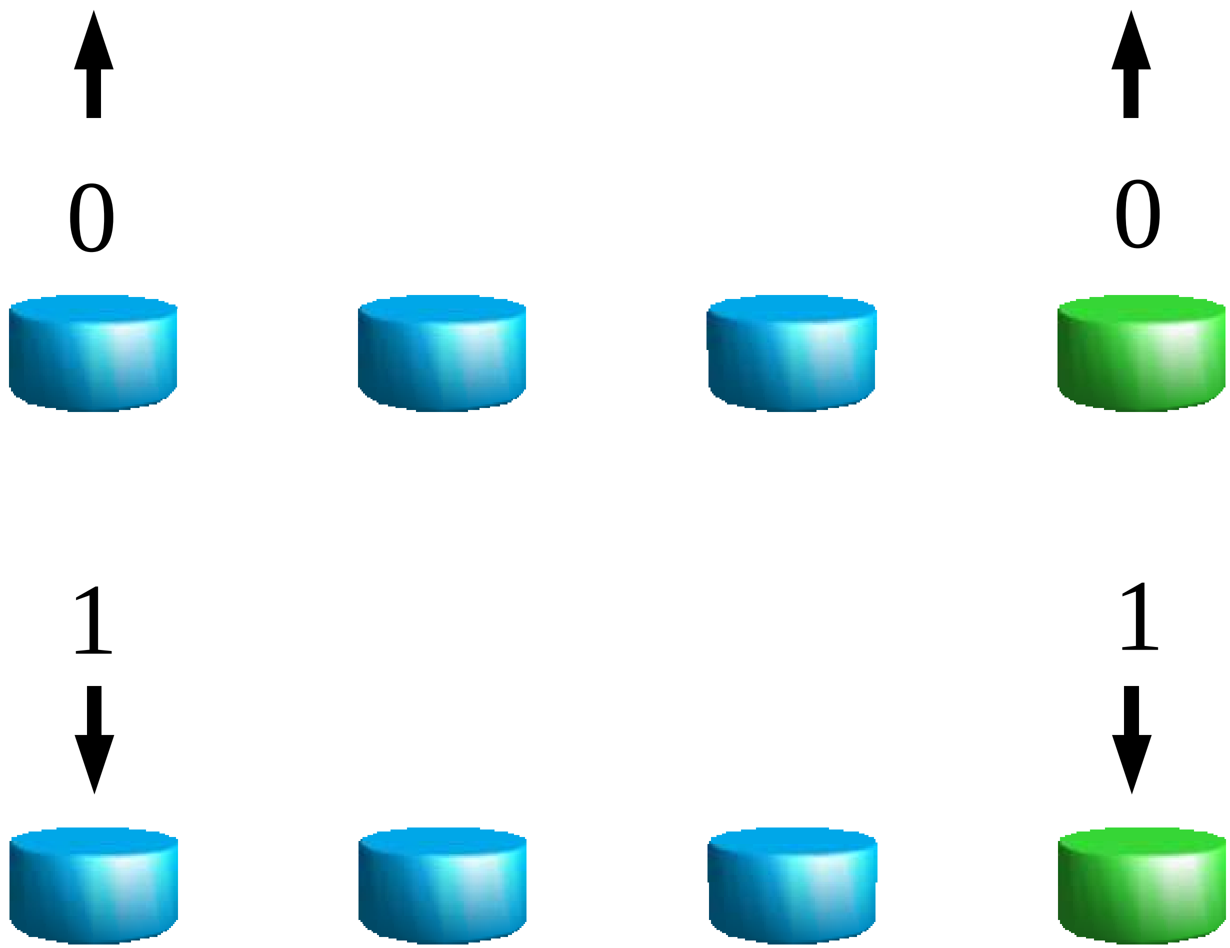
Single-block *Writes*

Modifying **a single block is harder**

Must maintain parity invariant for stripe

- **Could read full stripe, modify block, store full stripe**
- **Cheaper to fetch old versions of data block, parity block**
  - **Change a block of 0's to a block of 1's?**
  - **Old condition:** $0 \oplus X \oplus Y = 0$
  - **New condition:** $1 \oplus X \oplus Y = 1$
    - **Every bit flip in data causes a bit flip in parity**
    - **Independent of what's in X and Y, so don't read them in**
  - **Four disk operations – two read, two write**

Single-block Write



Parity Disk is a "Hot Spot"

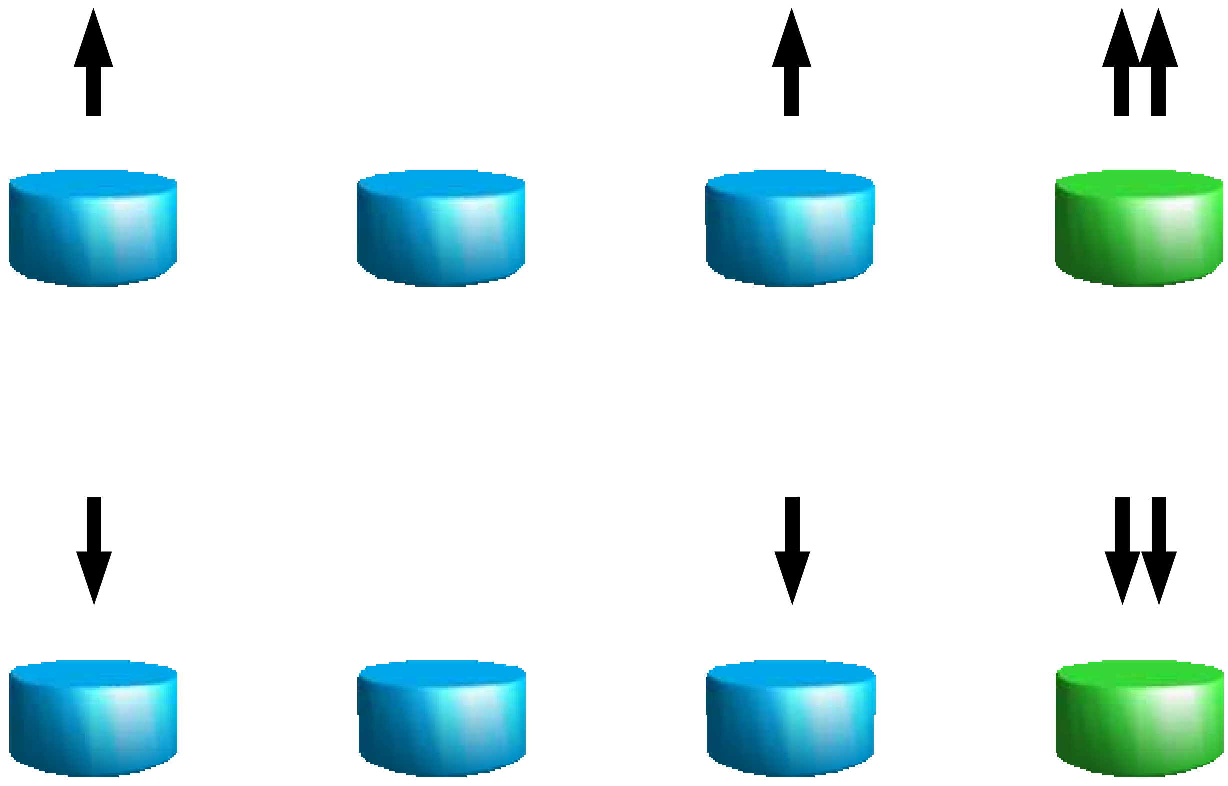
Single-block reads can happen in parallel

- **Each 1-block read affects only one disk**

Single-block writes *serialize*

- **Each 1-block write needs the parity disk**
  - **Twice!**

Sector-Write Hot Spot



RAID 4 – Summary

Like RAID 3

- Uses parity, relies on erasure signals from disks
- But unit = *block* instead of *bit*

Single-sector reads involve only 1 disk

- Can handle multiple single-sector reads in parallel

Single-sector writes: read, read, write, write!

Rarely used: parity disk is a *hot spot*

- Ok if all writes are large – NetApp WAFL file system writes in a log-like fashion

RAID 5

RAID 4, distribute parity among disks

No more “parity disk hot spot”

- Each small write still reads 2 disks, writes 2 disks
- But if you're lucky the sets don't intersect

Frequently used

Other fun flavors

RAID 6 – handle two simultaneous drive failures

- 2/N of overall space is used for parity instead of 1/N
- Implement as "two-dimensional parity", "P+Q" with Reed-Solomon ECC, or NetApp RAID-DP
- Depending on scheme, small writes require six I/O's vs. 4 for RAID 5!

RAID 0+1

- Stripe data across half of your disks
- Use the other half to mirror the first half
- Characteristics
  - RAID 0 lets you scale to arbitrary size
  - Mirroring gives you safety, good read performance
  - "Imaging applications"

Other fun flavors

RAID 1.5, 7, 10, DP, S, ...

- Many other varieties...
- Mixture of esoteric, single-vendor, non-standard terminology, marketing stunt, ...

Applications

RAID 0

- Temporary storage / swapping
- Not reliable!

RAID 1

- Simple to explain, reasonable performance, expensive
- Traditional high-reliability applications (banking)

RAID 5

- Cheap reliability for large on-line storage
- AFS servers (*your* AFS servers!)

Are failures independent?

With RAID (1-5) disk failures are "ok"

***Array* failures are never ok**

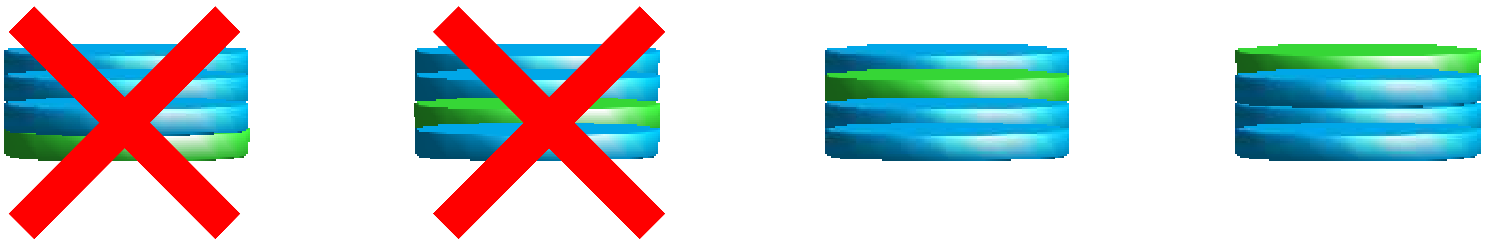
- **Cause: "Too many" disk failures "too soon"**
- **Result: No longer possible to XOR back to original data**
- **Hope your backup tapes are good...**
- **...and your backup system is tape-drive-parallel!**

Luckily, multi-disk failures are "rare"

- **After all, disk failures are "independently distributed"...**

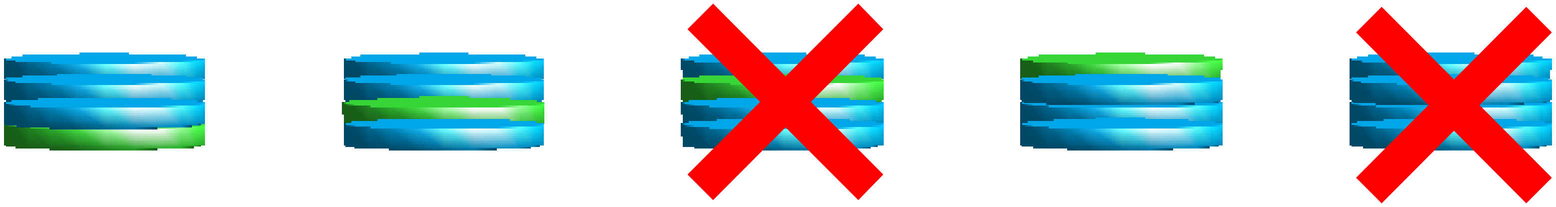
#insert <quad-failure.story>

Are failures independent?



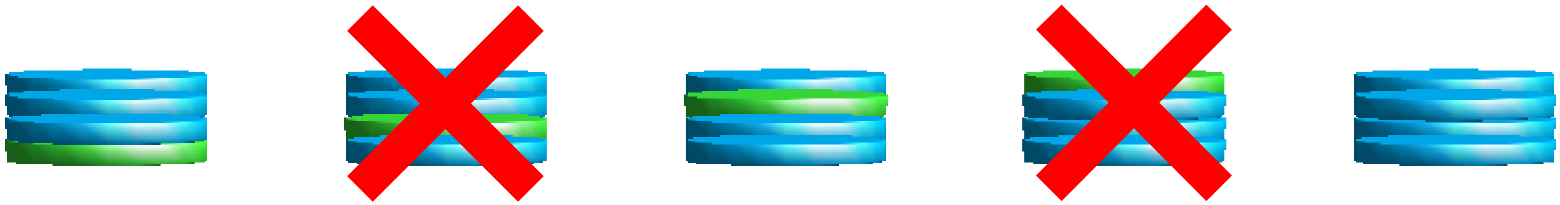
[See Hint 1]

Are failures independent?



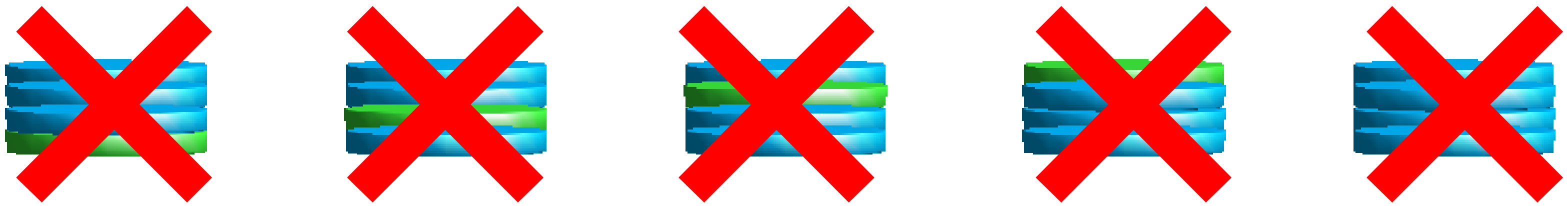
[See Hint 2]

Are failures independent?



[See Hint 3]

Are failures independent?



[See Hint 4]

Hints

Hint 1: 2 disks per IDE cable (or: controller failure)

Hint 2: If you never use it, does it still work?

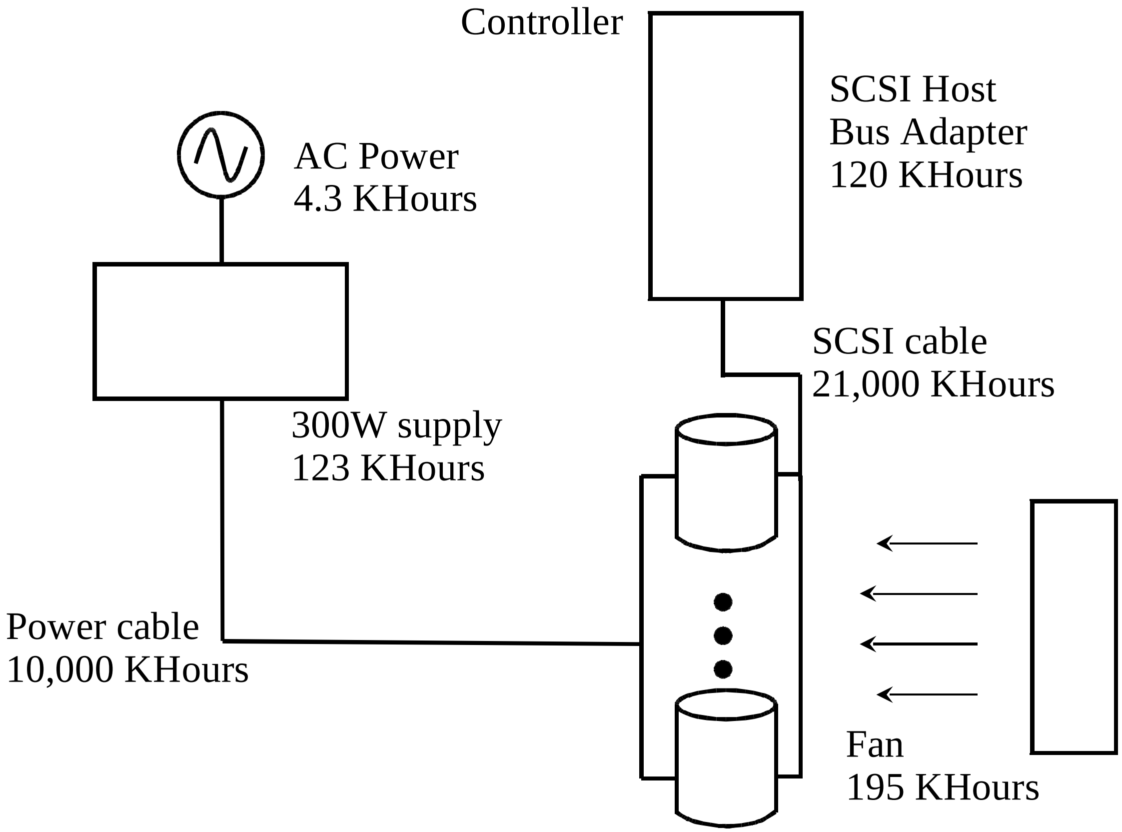
Hint 3: Some days are bad days

Hint 4: “Tunguska impact event” (1908, Russia)

Arrays Contain Support Hardware

Array includes many non-disk components

- Big threat: problems with external power
- Combined effects of non-disk components rival disk failures

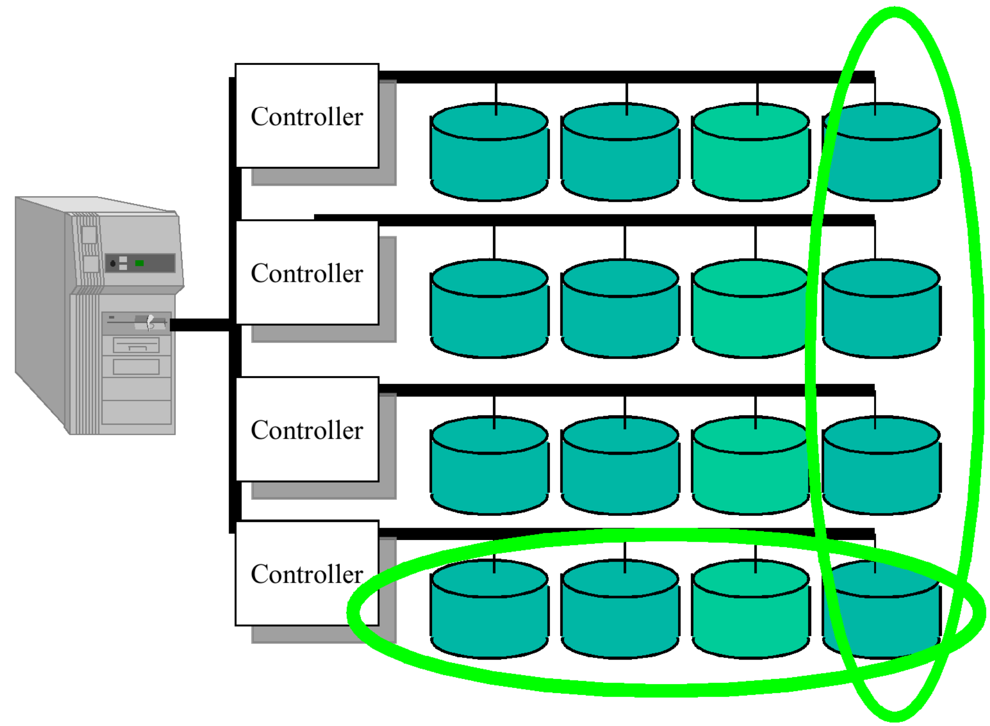


Schulze, Compcon, 1989

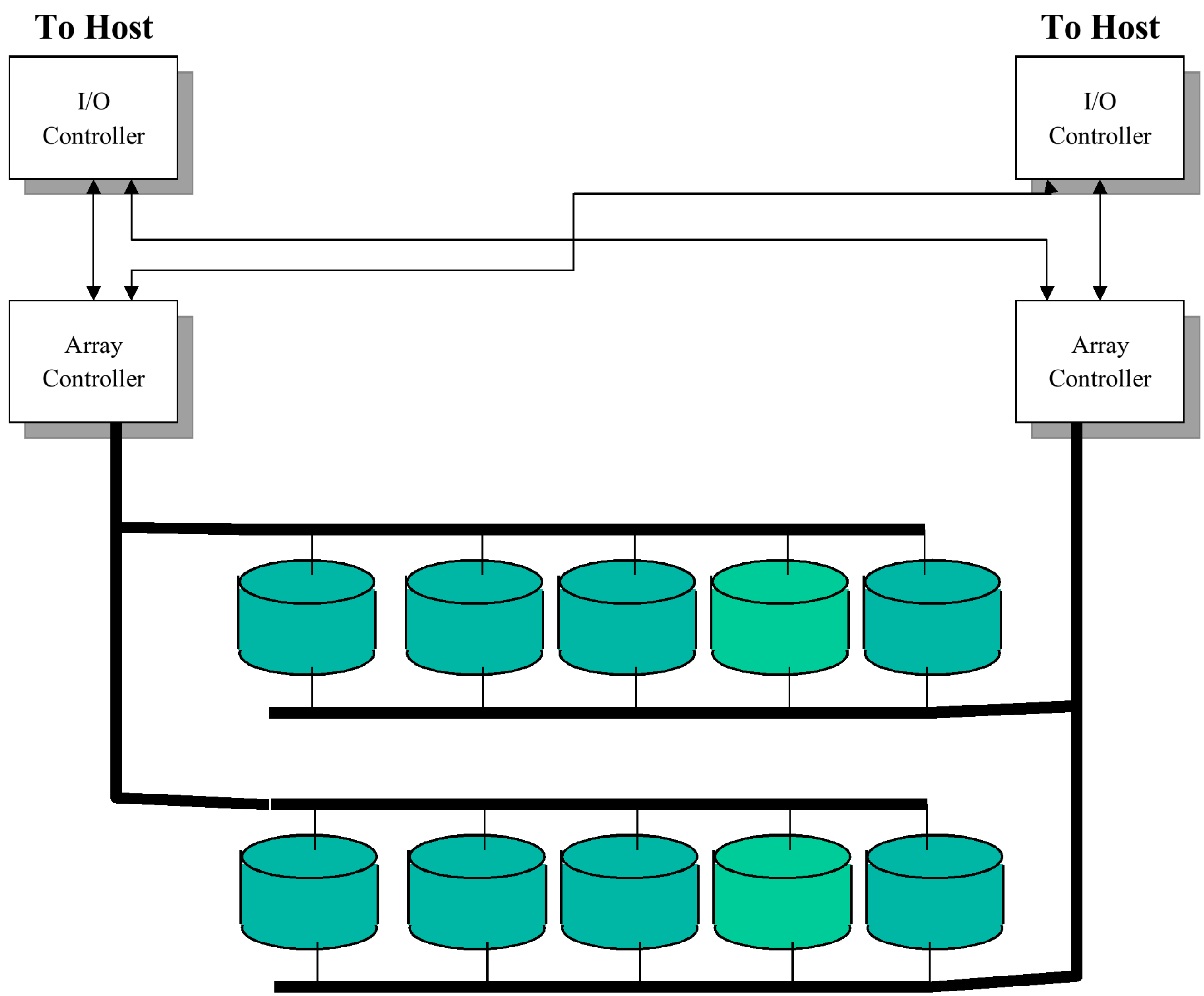
"Orthogonal" Redundancy

Alternative 4-disk array setups

- A) one "equation" per controller
- B) each "equation" across all the controllers
  - Tolerates lose of "string"



Dual Paths (Data and Power)



Summary

Need more disks!

- **More space, lower latency, more throughput**

Cannot **tolerate 1/N reliability**

Store information carefully and redundantly

Lots of variations on a common theme

You should understand RAID 0, 1, 5

RAID Papers

1988: Patterson, Gibson, Katz: A Case for Redundant Arrays of Inexpensive Disks (RAID)

- www.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf

1990: Chervenak, Performance Measurements of the First RAID Prototype

- www.isi.edu/~annc/papers/masters.ps
- This is a carefully-told "performance leaks away" story

1995: Tutorial on RAID

- http://www.pdl.cmu.edu/RAIDtutorial/Sigarch95.pdf

RAID Papers

2004: Corbett et al., Row-Diagonal Parity for Double Disk Failure Correction

- **www.usenix.org/events/fast04/tech/corbett/corbett.pdf**

2009: Plank et al., A Performance Evaluation and Examination of Open-Source Erasure Coding Libraries for Storage

- **www.usenix.org/events/fast09/tech/full_papers/plank/**

Other Papers

U.S. Patent 4,092,732

- "System for recovering data stored in failed memory unit," Norman Ken Ouchi, 1978 (assigned to IBM).
- http://www.google.com/patents?vid=USPAT4092732

Dispersed Concentration: Industry Location and Globalization in Hard Disk Drives

- David McKendrick, UCSD Info. Storage Industry Center
- Some history of disk market (1956-1998)
- isic.ucsd.edu/papers/dispersedconcentration/index.shtml