Dependability

Administrivia

- Check-Ins Round 2 Continue
  - We have left sign-up open just in case
- Project 4 is due 4/26 (Next Week)
  - We are relaxing the speedup requirements — more info on Piazza and more to be released soon
- Reminder: New Office Hour Queue Policy
  - Ticket will be marked as resolved if less than 1 hour old
- Final Exam on May 13
  - Same proctoring routine as midterm exam
  - Allowed unlimited handwritten notes + Green Sheet Reference

Great Idea #6: Dependability via Redundancy

- Applies to everything from data centers to memory
  - Redundant data centers so that can lose 1 datacenter but Internet service stays online
  - Redundant routes so can lose nodes but Internet doesn't fail
    - Or at least can recover quickly…
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ ECC Memory)

Dependability Corollary: Fault Detection

- The ability to determine that ***something*** is wrong is often the key to redundancy
  - "Work correctly or fail" is far easier to deal with than "May work incorrectly on failure"
- Error detection is generally a necessary prerequisite to error correction
  - And errors aren't just errors, but can be potential avenues for exploitation!

Dependability via Redundancy: Time vs. Space

- ***Spatial Redundancy*** – replicated data or extra information or hardware to handle hard and soft (transient) failures
- ***Temporal Redundancy*** – redundancy in time (retry) to handle soft (transient) failures
  - "~~Insanity~~ overcoming soft failures is repeatedly doing the same thing and expecting different results"

Dependability Measures

- Reliability: Mean Time To Failure (***MTTF***)
- Service interruption: Mean Time To Repair (***MTTR***)
- Mean time between failures (***MTBF***)
  - MTBF = MTTF + MTTR
- Availability = ***MTTF*** / (***MTTF*** + ***MTTR***)
- Improving Availability
  - Increase ***MTTF***: More reliable hardware/software + Fault Tolerance
  - Reduce ***MTTR***: improved tools and processes for diagnosis and repair

Availability Measures

- Availability = MTTF / (MTTF + MTTR) as %
  - MTTF, MTBF usually measured in hours
- Since we hope things are rarely down, shorthand is "number of 9s of availability per year"
- 1 nine: 90% => 36 days of repair/year
  - Airbears Reliability?
- 2 nines: 99% => 3.6 days of repair/year
- 3 nines: 99.9% => 526 minutes of repair/year
- 4 nines: 99.99% => 53 minutes of repair/year
- 5 nines: 99.999% => 5 minutes of repair/year
  - And serious $$$ to do

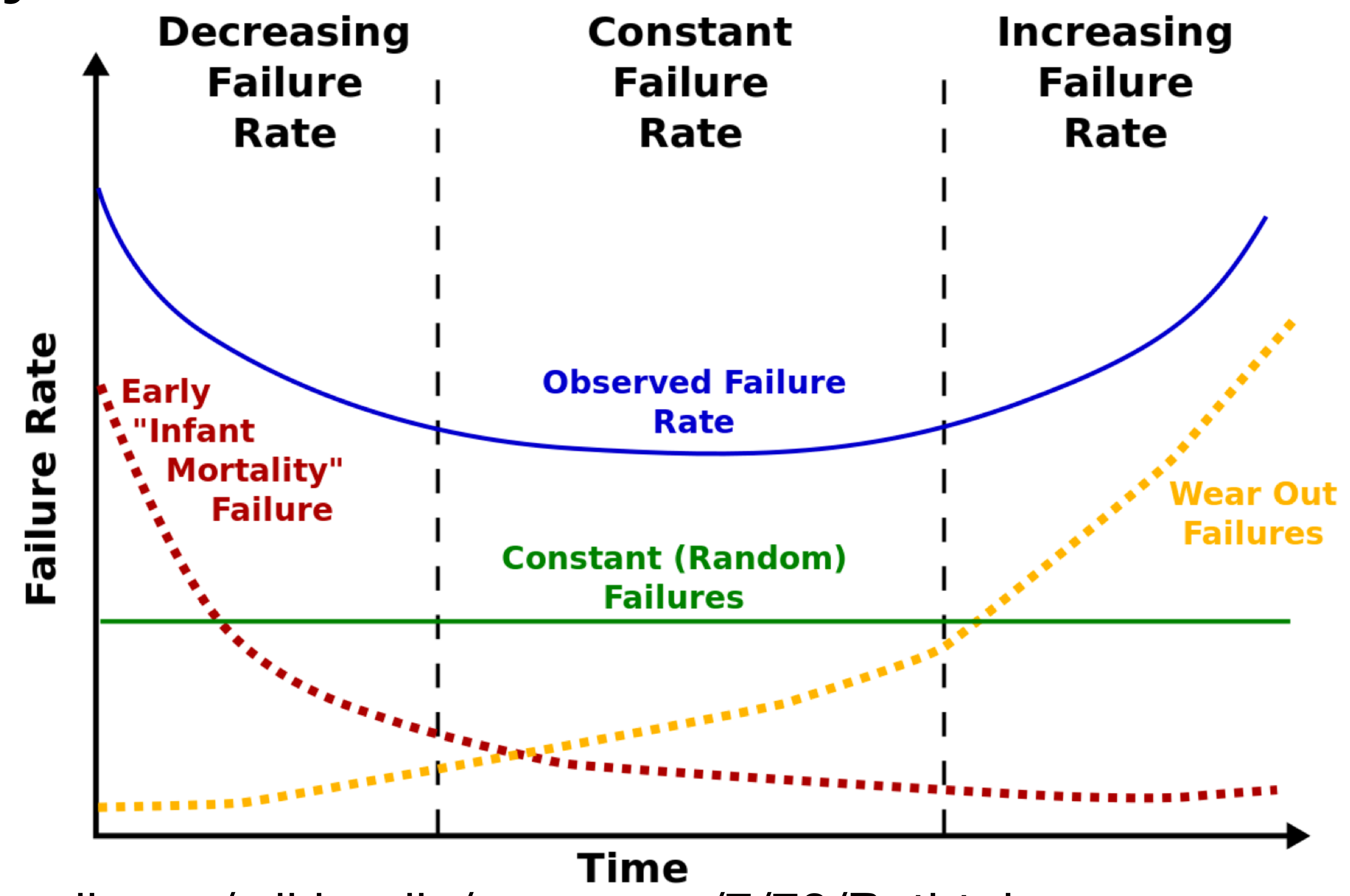
Reliability Measures

- Another is average number of failures per year: ***Annualized Failure Rate (AFR)***
  - E.g., 1000 disks with 100,000 hour MTTF
  - 365 days * 24 hours = 8760 hours
  - (1000 disks * 8760 hrs/year) / 100,000 = 87.6 failed disks ***per year*** on average
  - 87.6/1000 = 8.76% annual failure rate
- Google's 2007 study* found that actual AFRs for individual drives ranged from 1.7% for first year drives to over 8.6% for three-year old drives

*research.**google**.com/archive/disk_failures.pdf*

The "Bathtub Curve"

- Often failures follow the "bathtub curve"
- Brand new devices may fail
- Old devices fail
- Random failure in between



https://upload.wikimedia.org/wikipedia/commons/7/78/Bathtub_curve.svg

Dependability Design Principle

- Design Principle: No single points of failure
  - “Chain is only as strong as its weakest link”
- Dependability behaves like speedup of Amdahl’s Law
  - Doesn’t matter how dependable you make one portion of system
  - Dependability limited by part you do not improve

Error Detection/Correction Codes

- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - "***Soft***" errors occur occasionally when cells are struck by alpha particles or other environmental upsets
  - "***Hard***" errors can occur when chips permanently fail
  - Problem gets worse as memories get denser and larger
- Memories protected against failures with EDC/ECC
- Extra bits are added to each data-word
  - Used to detect and/or correct faults in the memory system
  - Each data word value mapped to unique code word
  - A fault changes valid code word to invalid one, which can be detected

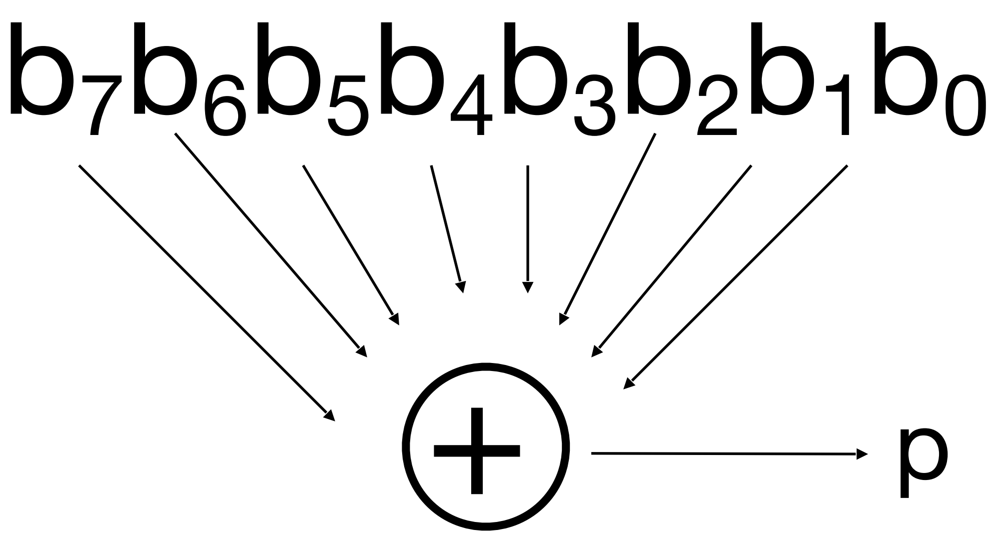
Block Code Principles

- Hamming distance = difference in # of bits
- p = 011011, q = 001111, Ham. distance (p,q) = 2
- p = 011011,
  q = 110001,
  distance (p,q) = ?
- Can think of extra bits as creating a code with the data
- What if minimum distance between members of code is 2 and get a 1-bit error?

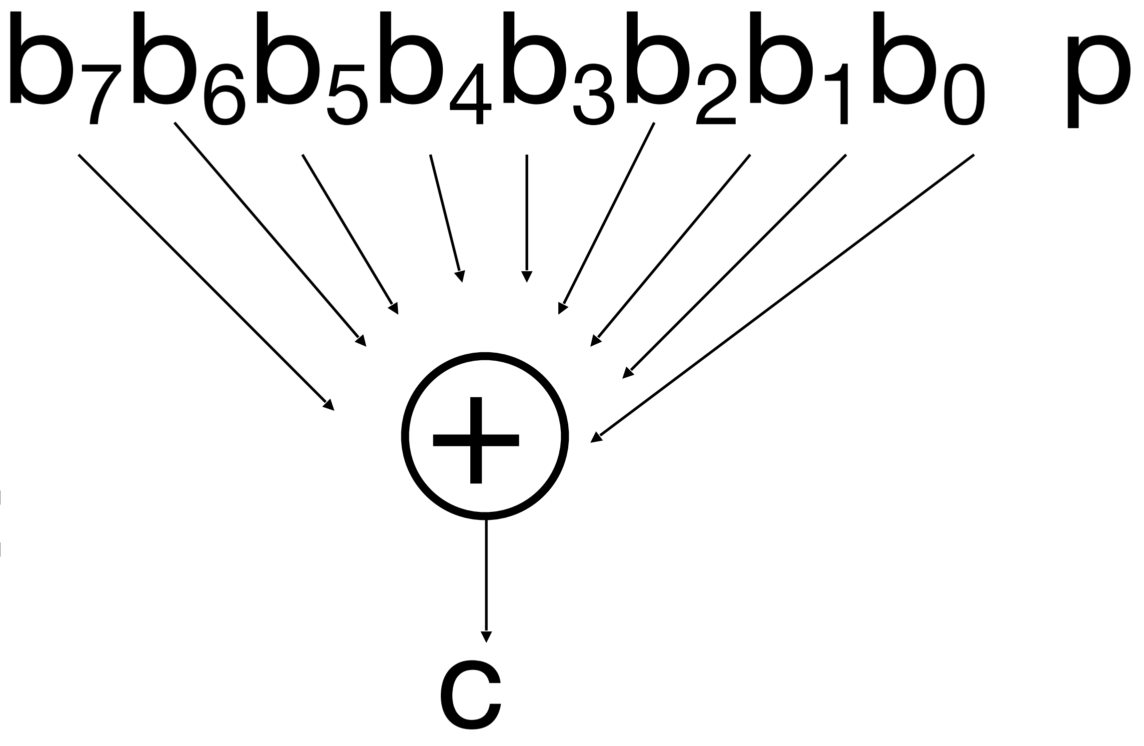
Richard Hamming
Turing Award Winner

Parity: Simple Error-Detection Coding

- Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have *even parity*:



- Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).



- Minimum Hamming distance of parity code is 2
- A non-zero parity check indicates an error occurred:
  - 2 errors (on different bits) are not detected
  - nor any even number of errors, just odd numbers of errors are detected

Parity Example

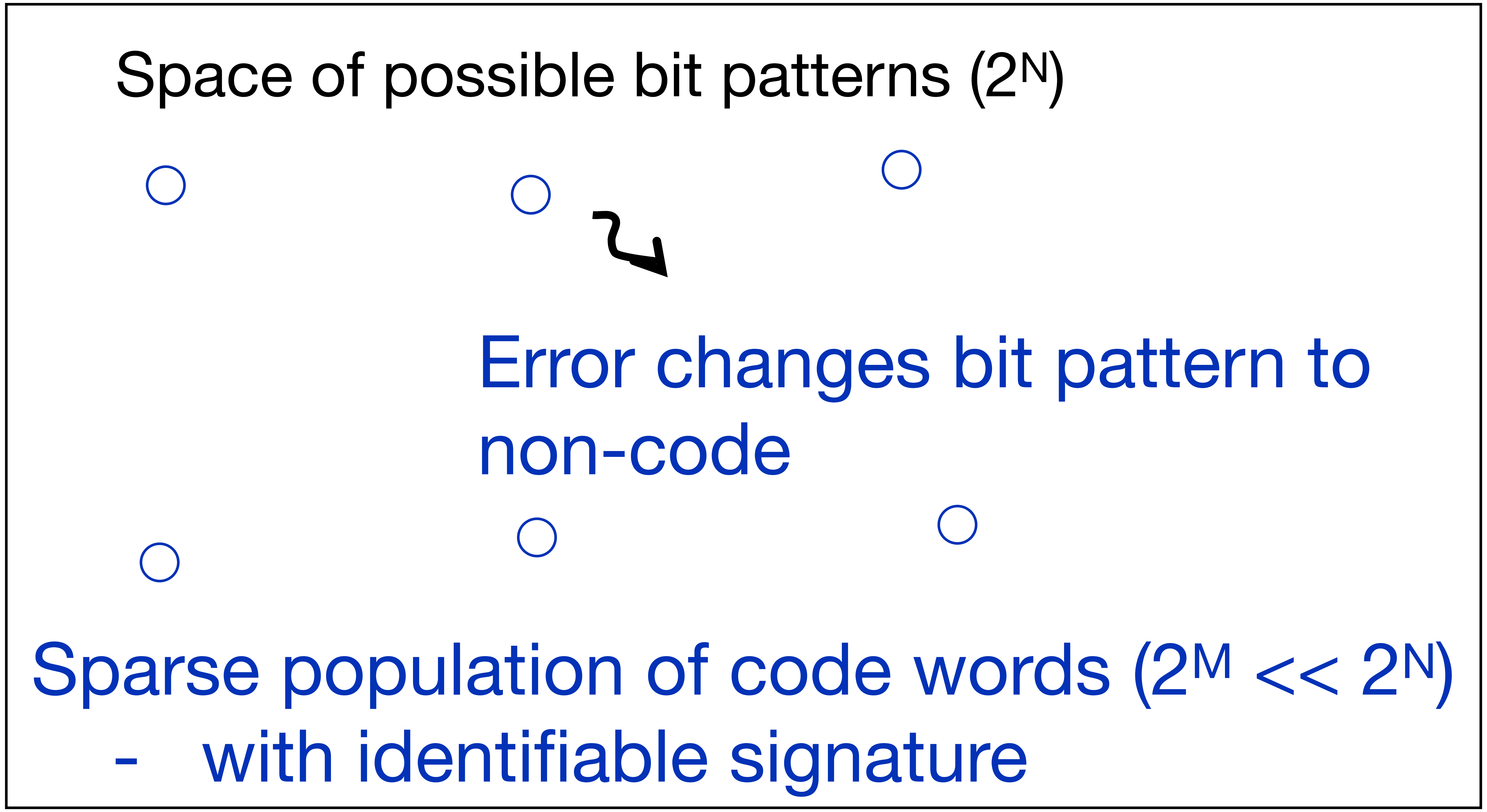
- Data 0101 0101
- 4 ones, even parity now
- Write to memory:
  0101 0101 0
  to keep parity even
- Data 0101 0111
- 5 ones, odd parity now
- Write to memory:
  0101 0111 1
  to make parity even
- Read from memory
  0101 0101 0
- 4 ones => even parity, so no error
- Read from memory
  1101 0101 0
- 5 ones => odd parity, so error
- What if error in parity bit?

Suppose Want to Correct 1 Error?

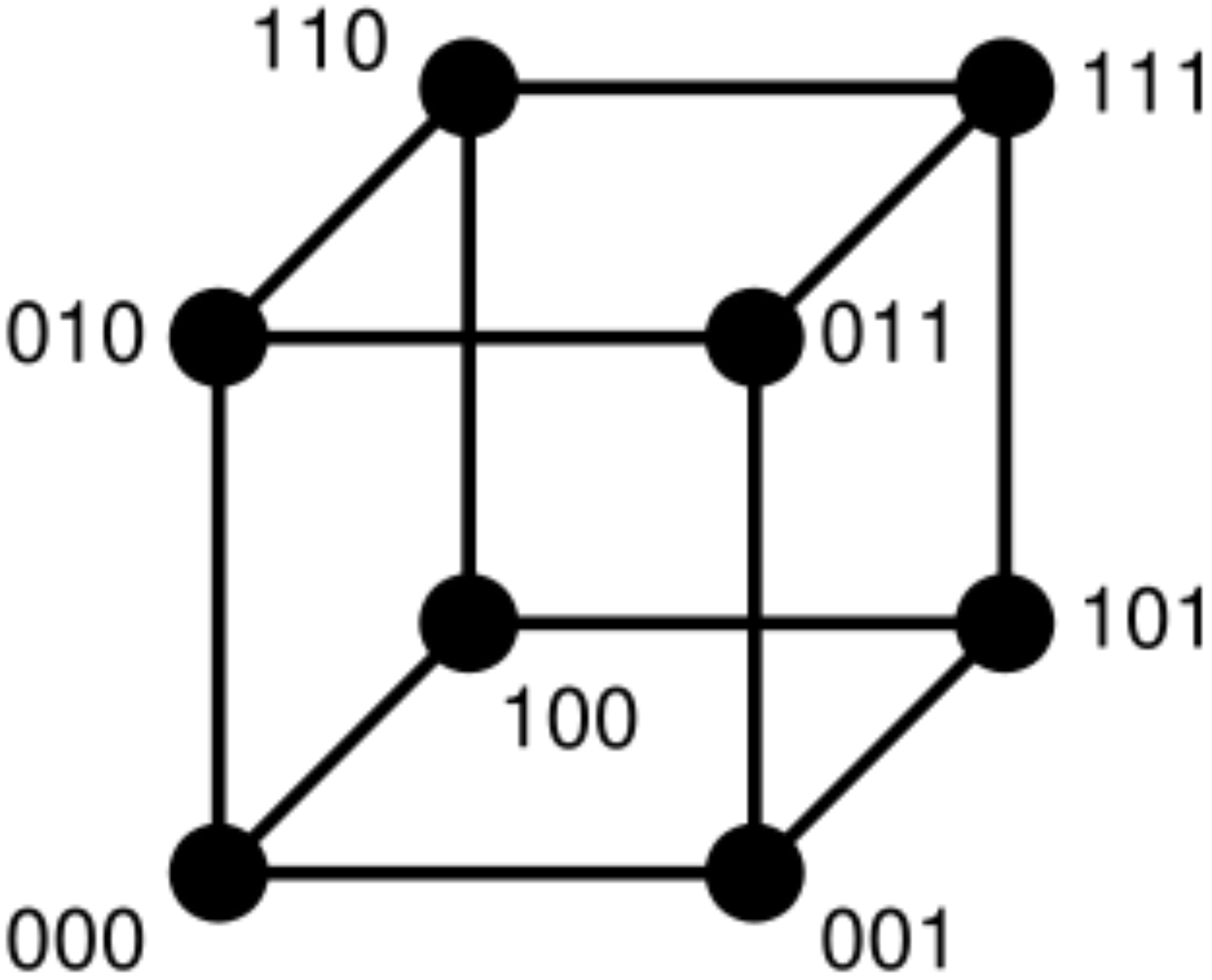
- Richard Hamming came up with simple to understand mapping to allow Error Correction at minimum distance of 3
  - Single error correction, double error detection
- Called "Hamming ECC"
  - Worked weekends on relay computer with unreliable card reader, frustrated with manual restarting
  - Got interested in error correction; published 1950
  - R. W. Hamming, "Error Detecting and Correcting Codes," The Bell System Technical Journal, Vol. XXVI, No 2 (April 1950) pp 147-160.

Detecting/Correcting Code Concept

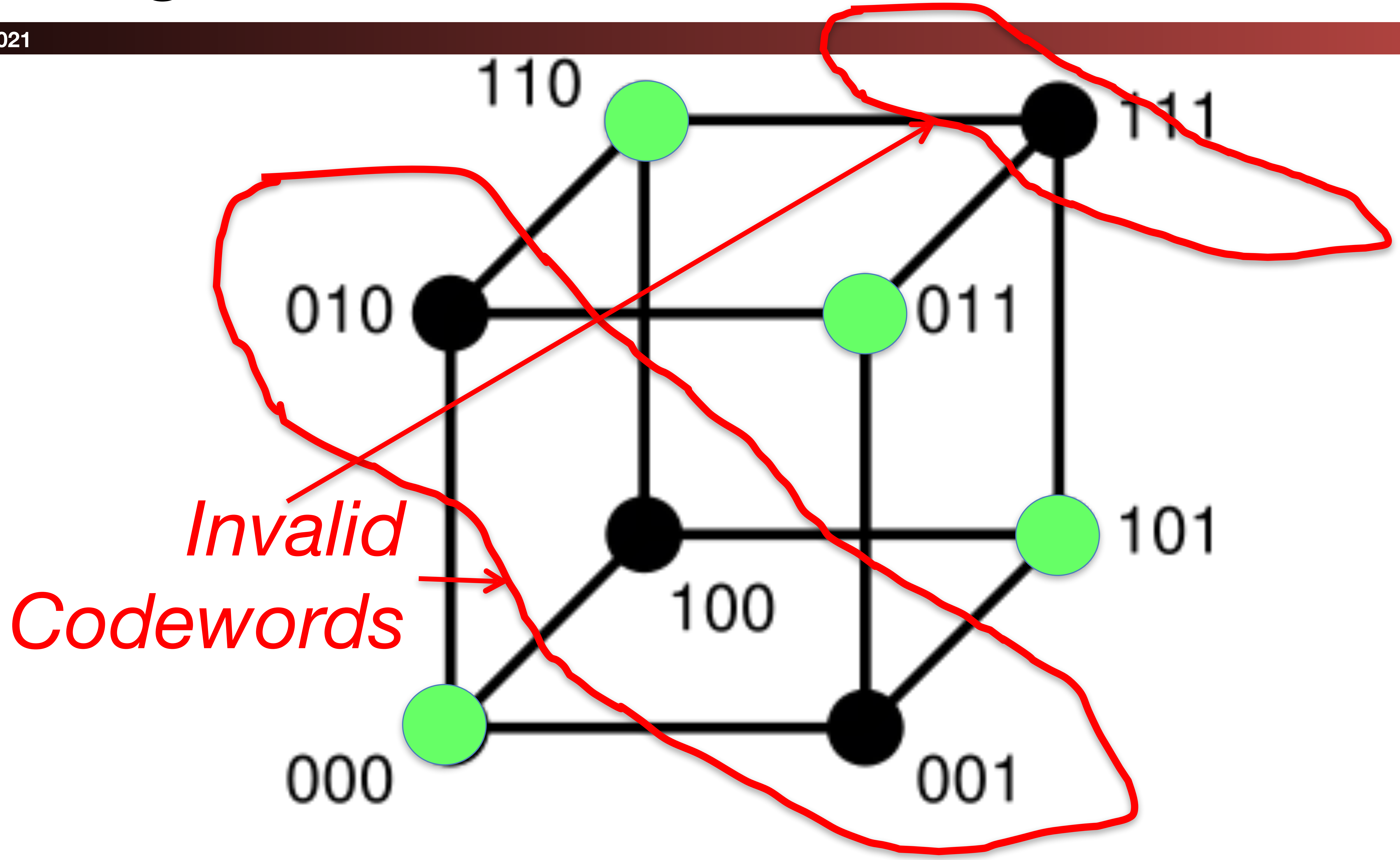
- Detection: bit pattern fails codeword check
- Correction: map to nearest valid code word



Hamming Distance: 8 code words



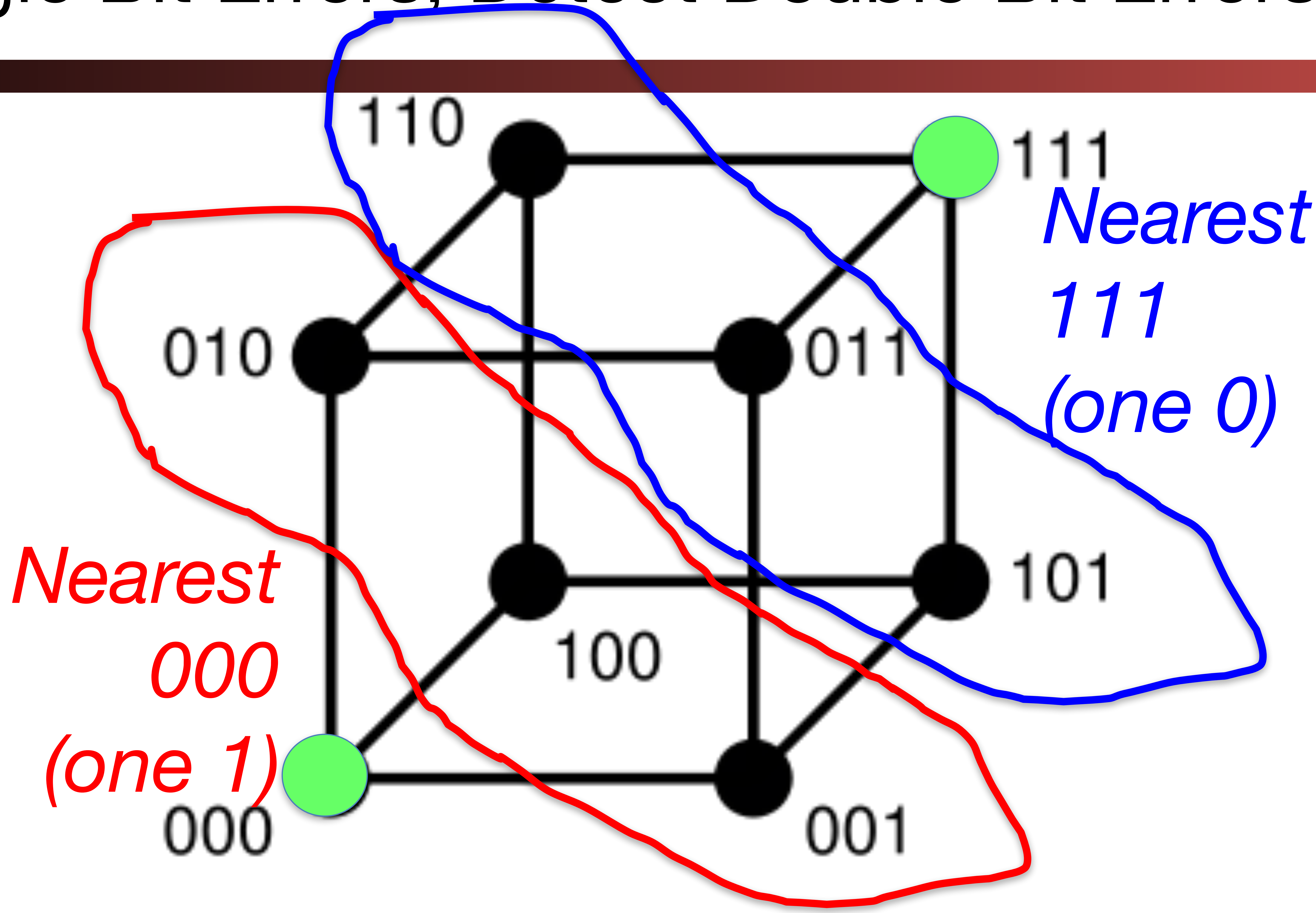
Hamming Distance 2: Detection
Detect Single Bit Errors



- No 1 bit error goes to another valid codeword
- ½ codewords are valid
  - This is parity

Hamming Distance 3: Correction

Correct Single Bit Errors, Detect Double Bit Errors



- No 2 bit error goes to another valid codeword; 1 bit error near 1/4 codewords are valid

Graphic of Hamming Code

- http://en.wikipedia.org/wiki/Hamming_code

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Hamming ECC

Set parity bits to create even parity for each group

- A byte of data: 10011010
- Create the coded word, leaving spaces for the parity bits:
- ```
  _ _ 1 _ 0 0 1 _ 1 0 1 0
  1 2 3 4 5 6 7 8 9 a b c - bit position
  ```
- Calculate the parity bits
  ```

Hamming ECC

- Position 1 checks bits **1,3,5,7,9,11**:
  `? _ 1 _ 0 0 1 _ 1 0 1 0`. set position 1 to a <u>0</u>:
- Position 2 checks bits **2,3,6,7,10,11**:
  `0 ? 1 _ 0 0 1 _ 1 0 1 0`. set position 2 to a <u>1</u>:
- Position 4 checks bits **4,5,6,7,12**:
  `0 1 1 ? 0 0 1 _ 1 0 1 0`. set position 4 to a <u>1</u>:
- Position 8 checks bits **8,9,10,11,12**:
  `0 1 1 1 0 0 1 ? 1 0 1 0`. set position 8 to a <u>0</u>:

Hamming ECC

- Final code word: `011100101010`
- Data word: `1 001 1010`

Hamming ECC Error Check

- Suppose we receive
  0111001011110

0 1 1 1 0 0 1 0 1 1 1 0

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Hamming ECC Error Check

- Suppose receive
`011100101110`

Hamming ECC Error Check

- Suppose receive

```
011100101110
0 1 0 1 1 1   √
 11  01  11   x-Parity 2 in error
   1001    0  √
        01110 x-Parity 8 in error
```

- *Implies position 8+2=10 is in error*

```
011100101110
```

Hamming ECC Error Correct

- Flip the incorrect bit …
  `011100101010`

Hamming ECC Error Correct

- Suppose we receive

```
0111001010010
0 1 0 1 1 1   √
 11  01  01   √
   1001     0 √
       01010  √
```

One Problem: Malicious "errors"

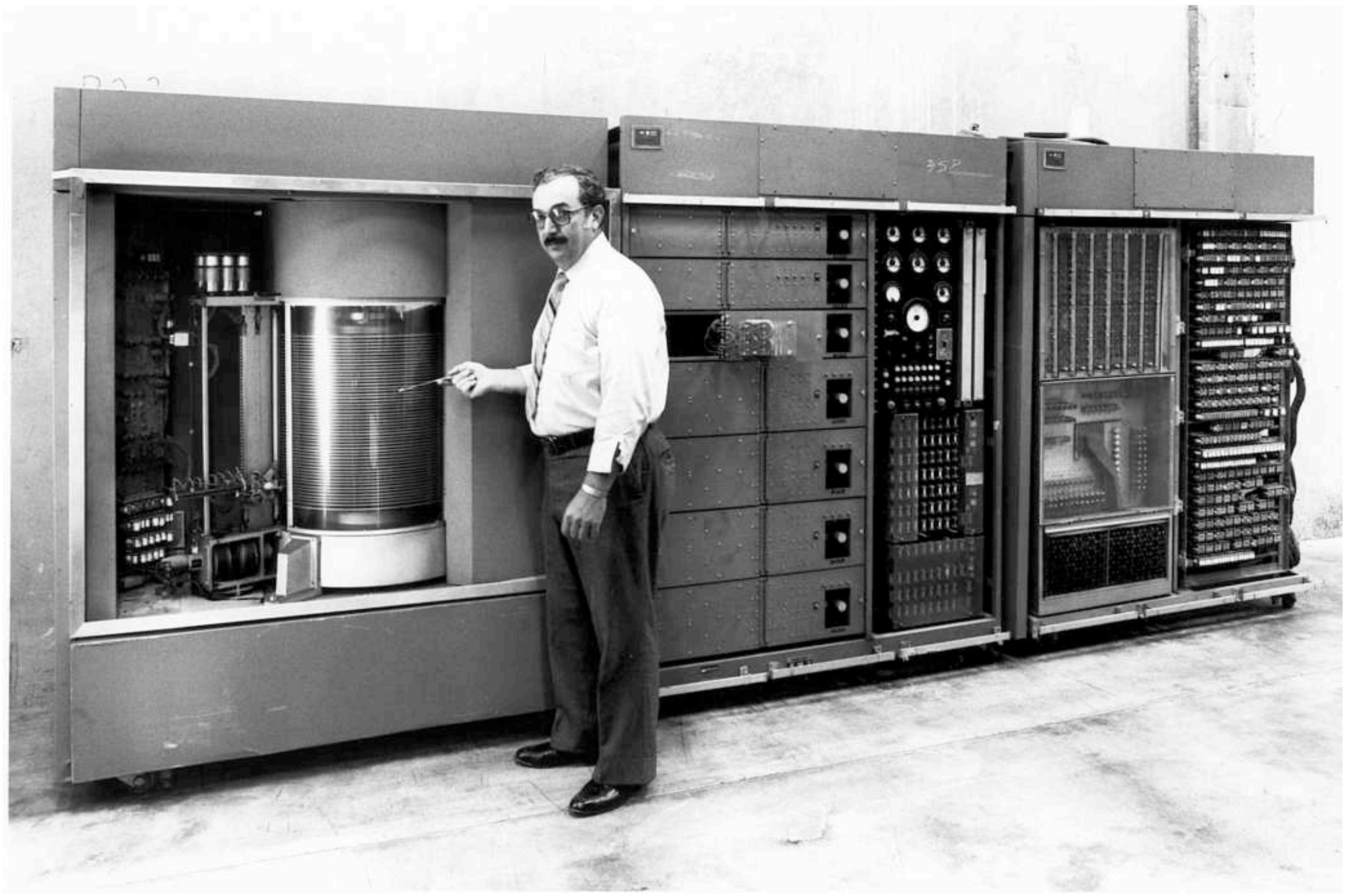
- Error Correcting Code and Error Detecting codes designed for ***random*** errors
- But sometimes you need to protect against ***deliberate*** errors
- Enter cryptographic hash functions
  - Designed to be nonreversible and unpredictable H(🐮)=hamburger
  - An attacker should not be able to change, add, or remove any bits without changing the hash output
    - For a 256b cryptographic hash function (e.g. SHA256), need to have 2^{128} items you are comparing before you have a reasonable possibility of a collision
  - This is also known as a "Message Digest": It does not correct errors but it can detect errors

RAID: Redundancy for Disk
So Why Worry About Disk At All?

- Spinning disk is still a critical technology
  - Although worse latency than SSD…
- Disk has equal or greater bandwidth and an order of magnitude better storage density (bits/cm^3) and cost density (bits/$)
- So when you need to store a petabyte or three…
  - You need to use disk, not SSDs
- Oh, and SSDs can fail too

Evolution of the Disk Drive

IBM 3390K, 1986



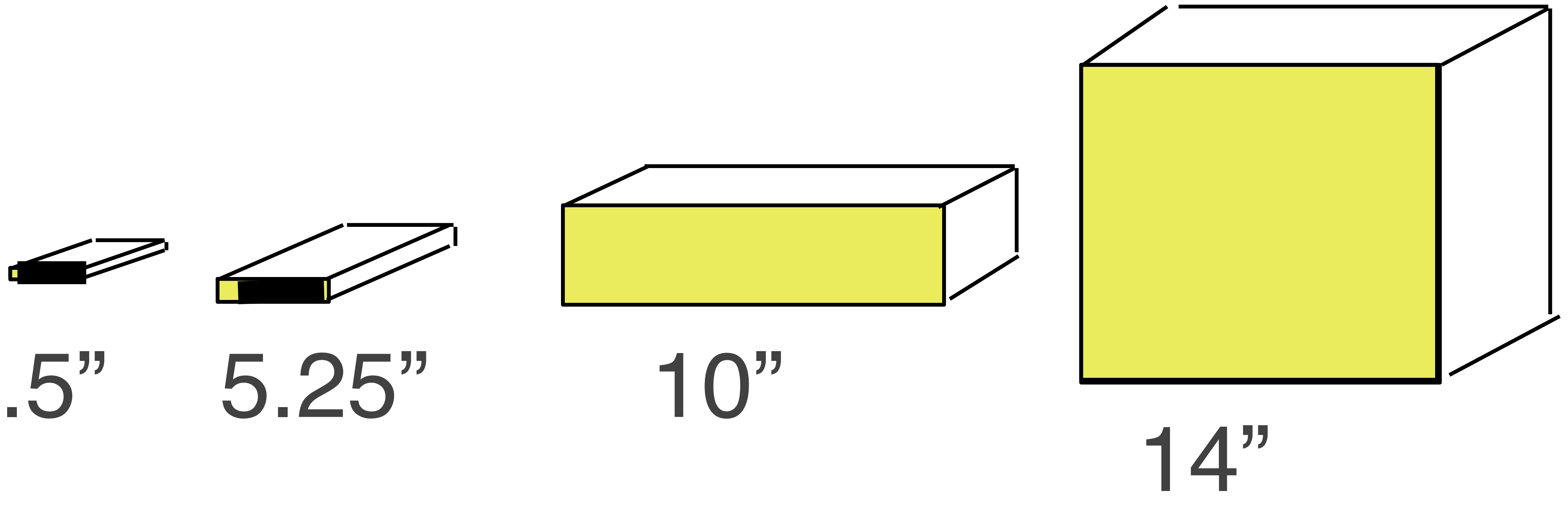
IBM RAMAC 305, 1956

Apple SCSI, 1986

Arrays of Small Disks

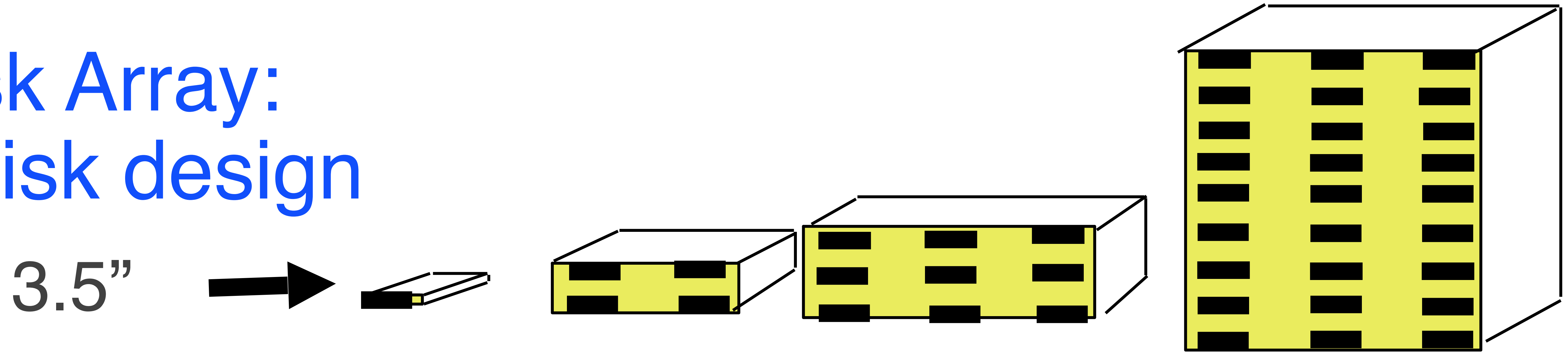
Can smaller disks be used to close gap in performance between disks and CPUs?

Conventional:
4 disk designs



Low End → High End

Disk Array:
1 disk design



Replace Small Number of Large Disks with Large Number of Small Disks! (1988 Disks)

	IBM 3390K	IBM 3.5" 0061	x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	105 MB/s	7X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	??? Hrs	
Cost	$250K	$2K	$150K	

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?

But MTTF goes through the roof…

- If 1 disk as MTTF of 50k hours…
  - 70 disks will have a MTTF of ~700 hours!!!
  - This is assuming failures are independent…
- But fortunately we know when failures occur!
  - Disks use a lot of CRC coding, so we don't have corrupted data, just no data
- We can have both "Soft" and "Hard" failures
  - Soft failure just the read is incorrect/failed, the disk is still good
  - Hard failures kill the disk, necessitating replacement
    - Most RAID setups are "Hot swap":
      Unplug the disk and put in a replacement while things are still going
    - Most modern RAID arrays also have "hot spares":
      An already installed disk that is used automatically if another disk fails.

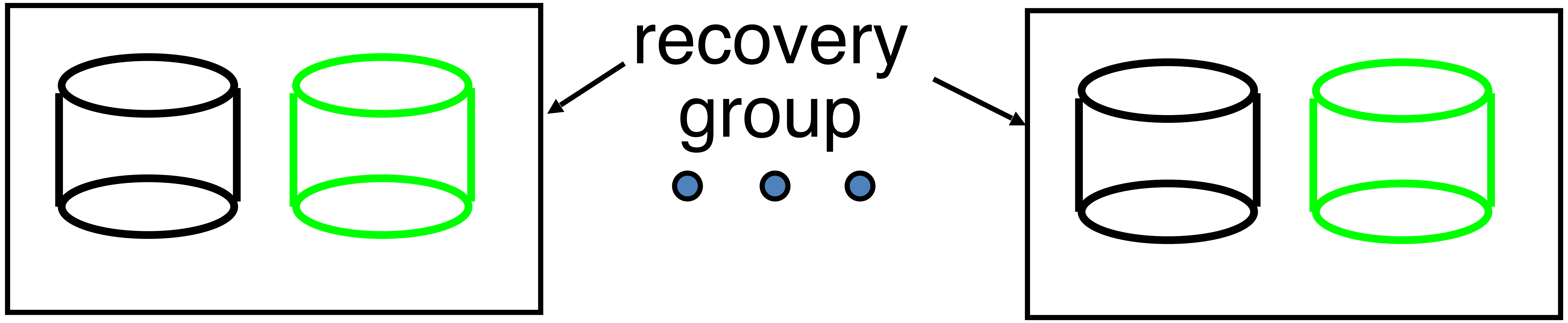
RAID: Redundant Arrays of (Inexpensive) Disks

- Files are "striped" across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info on writes

Raid 0: Striping

- "RAID 0" is not actually RAID (no redundancy)
  - It is simply spreading the data across multiple disks
- So, e.g, for 4 disks, address 0 is on disk 0, address 1 is on disk 1, address 2 is on disk 2, address 4 on disk 0...
- Improves bandwidth linearly
  - With 4 disks you have 4x the disk bandwidth
- Doesn't really help latency
  - Still have the individual disks seek and rotation time
- And well, failures happen...

Redundant Arrays of Inexpensive Disks
RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its "mirror"
    - Very high availability can be achieved
- Writes limited by single-disk speed
- Reads may be optimized

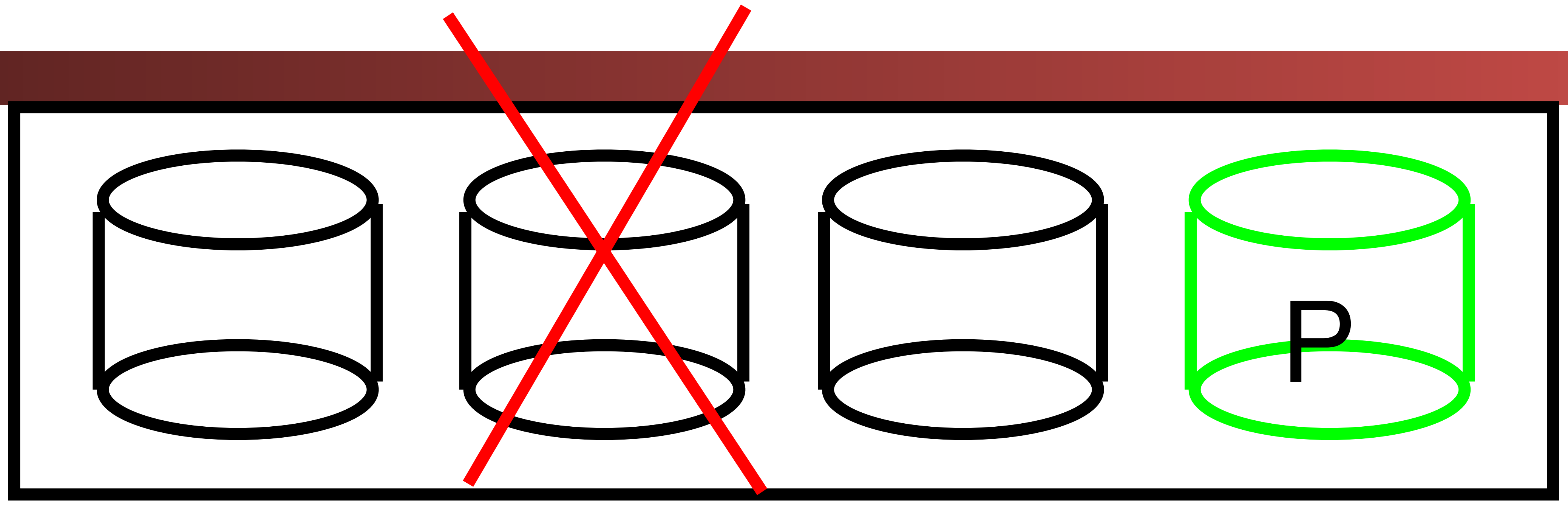
Most expensive solution: 100% capacity overhead

Redundant Array of Inexpensive Disks RAID 3: Parity Disk

10010011
11001101
10010011
. . .

logical record

Striped physical records

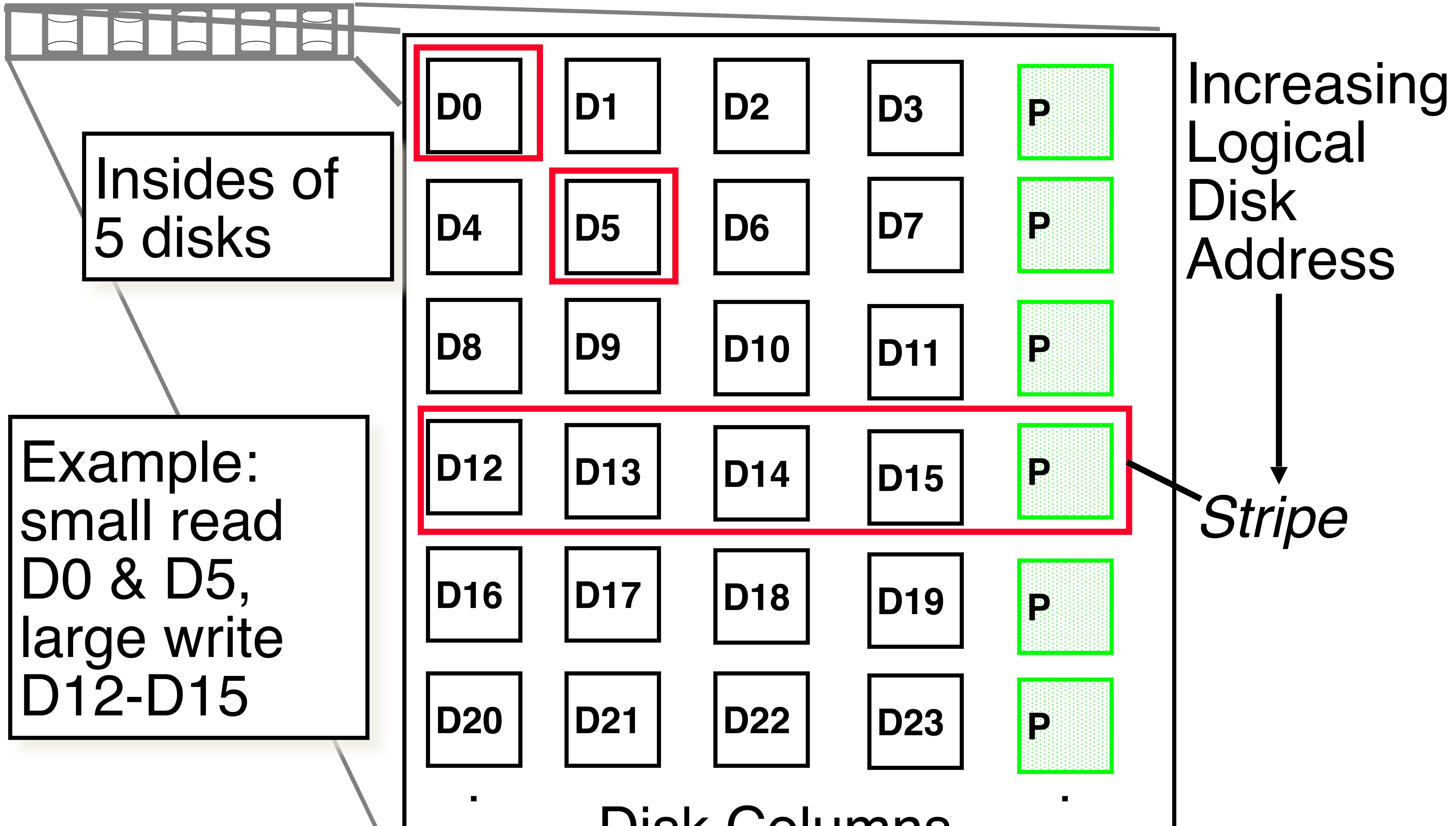


1	1	1	1
0	1	0	1
1	0	1	0
0	0	0	0
0	1	0	1
0	1	0	1
1	0	1	0
1	1	1	1

P contains sum of other disks per stripe mod 2 ("parity")

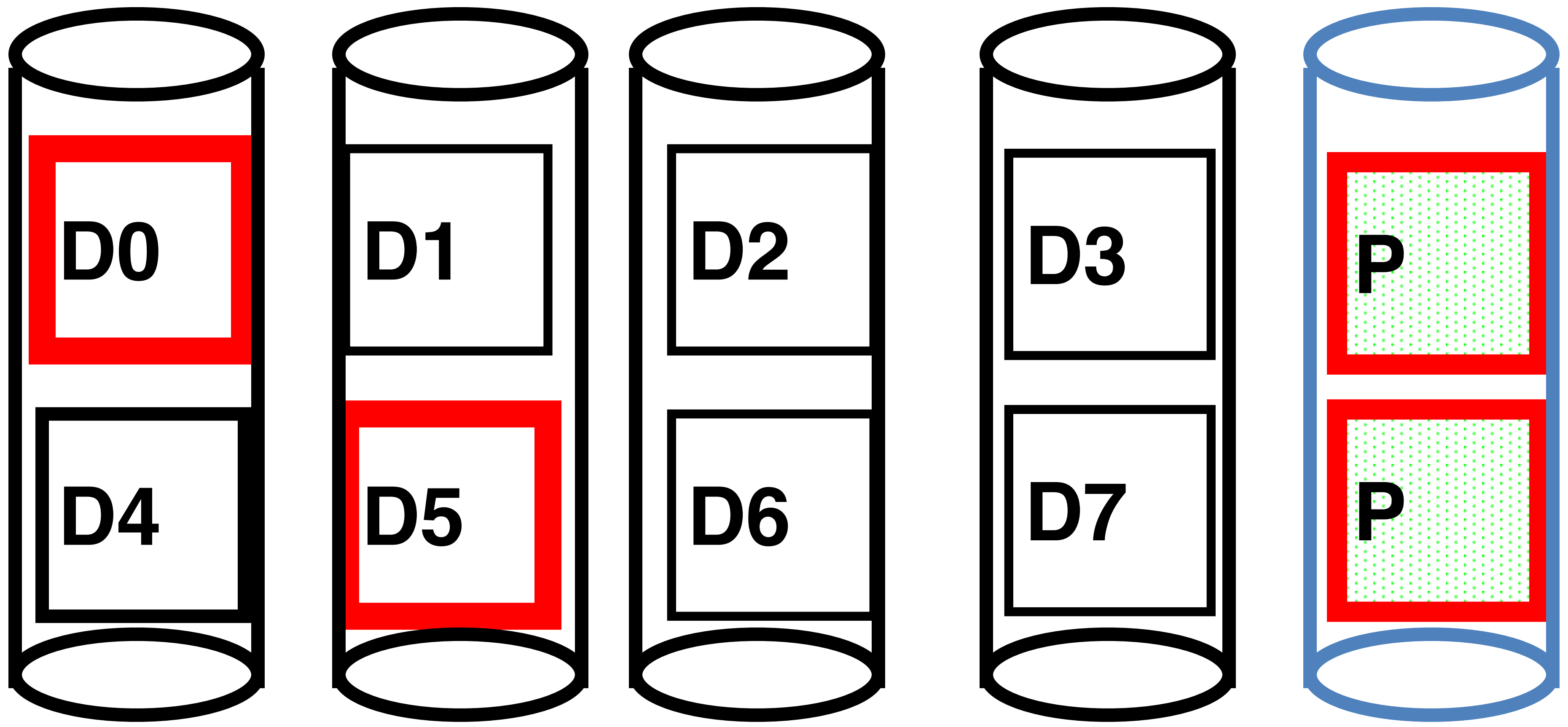
If disk fails, subtract P from sum of other disks to find missing information

Redundant Arrays of Inexpensive Disks RAID 4: High I/O Rate Parity

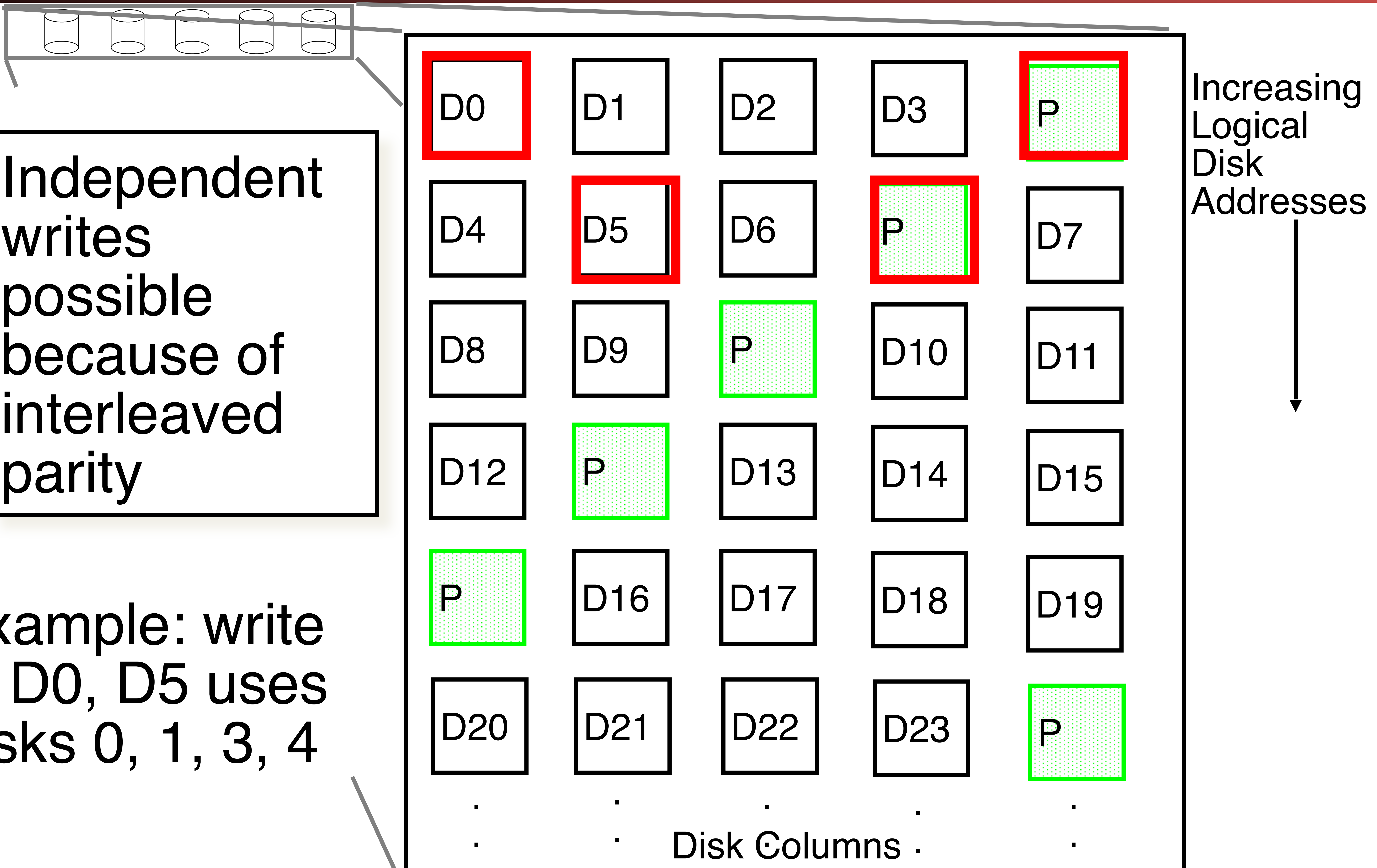


Inspiration for RAID 5

- RAID 4 works well for small reads
- Small writes (write to one disk):
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Small writes are limited by Parity Disk: Write to D0, D5 both also write to P disk



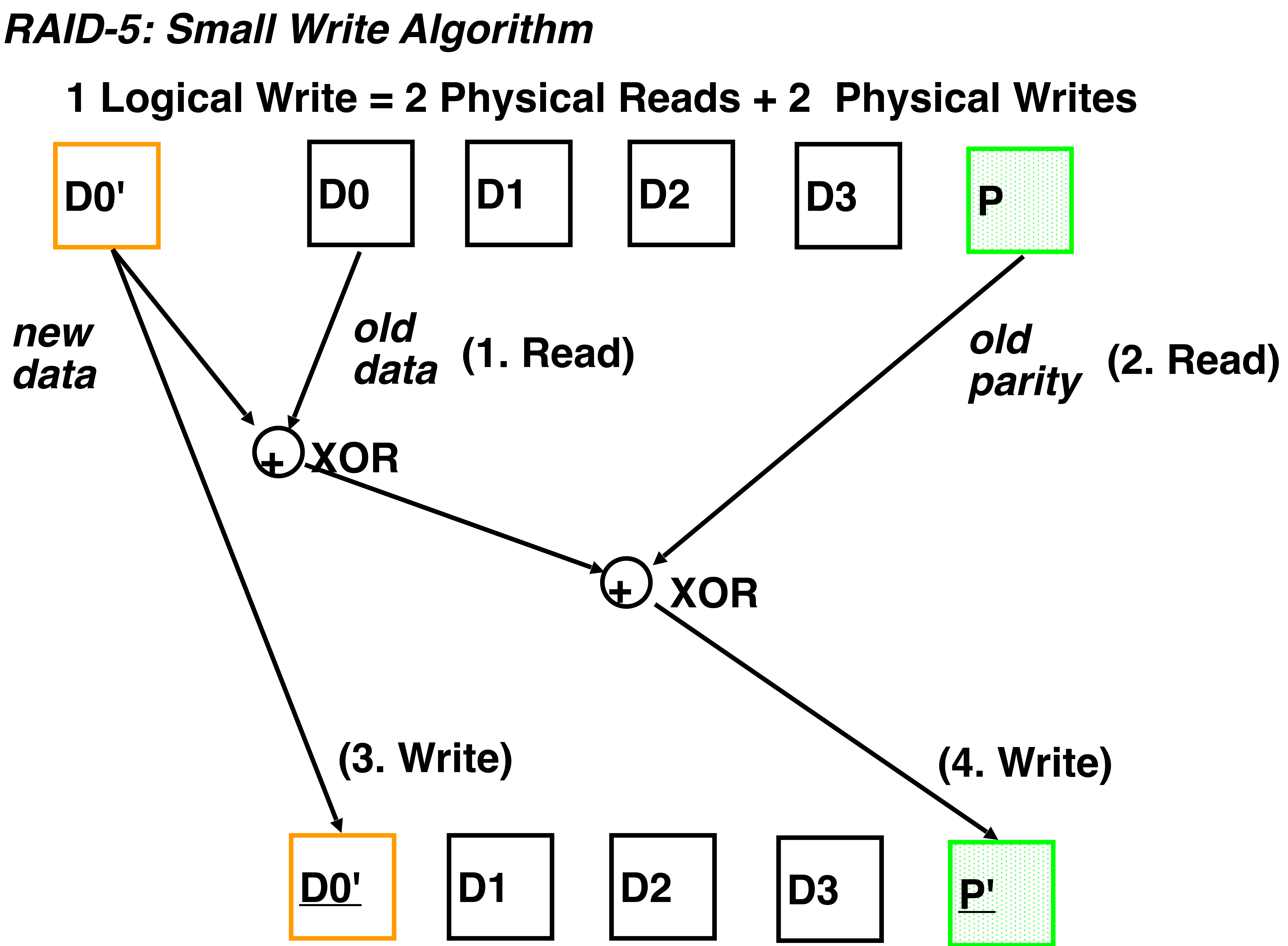
RAID 5: High I/O Rate Interleaved Parity



Independent writes possible because of interleaved parity

Example: write to D0, D5 uses disks 0, 1, 3, 4

Problems of Disk Arrays: Small Writes



RAID 6

- RAID 5 is no longer the “gold standard”
- Can experience 1 disk failure and continue operation
  - RAID array is in a “degraded” state
- But disk failures are not actually independent!
  - When one disk has failed, there’s a decent chance another will fail soon
- RAID 6: Add another parity block per stripe
  - Now 2 blocks per stripe rather than 1
  - Sacrifice capacity for increased redundancy
  - Now the array can tolerate **2** disk failures and continue operating

Berkeley's Role in Definition of RAID (December 1987)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

Google

Case for Raid

Scholar

About 138,000 results (**0.08** sec)

Articles

Legal documents

Any time

[BOOK] A **case** for redundant arrays of inexpensive disks (**RAID**)

DA Patterson, G Gibson, RH Katz - 1988 - dl.acm.org

Abstract Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disks (SLED) has grown rapidly, the performance improvement of SLED has been modest. ...

Cited by 2814 Related articles All 239 versions Cite More

Increasing performance of CPUs and memories will be squandered if not matched by a similar performance increase in I/O. While the capacity of Single Large Expensive Disk (SLED) has grown rapidly, the performance improvement of SLED has been modest. Redundant Arrays of Inexpensive Disks (RAID), based on the magnetic disk technology developed for personal computers, offers an attractive alternative to SLED, promising improvements of an order of magnitude in performance, reliability, power consumption, and scalability.

This paper introduces five levels of RAIDs, giving their relative cost/performance, and compares RAIDs to an IBM 3380 and a Fujitsu Super Eagle.

RAID Version 1

- RAID-I (1989)
  - Consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software

RAID Version 2

- 1990-1993
- Early Network Attached Storage (NAS) System running a Log Structured File System (LFS)
- Impact:
  - $25 Billion/year in 2002
  - Over $150 Billion in RAID device sold since 1990-2002
  - 200+ RAID companies (at the peak)
  - Software RAID a standard component of modern OSs

RAID Is Not Enough By Itself

- You don't just have one disk die...
  - You can have more die in a short period of time
  - Thank both the "bathtub curve" and common environmental conditions
- If you care about your data, RAID isn't sufficient
  - You need to also consider a separate backup solution
- A good practice in clusters/warehouse scale computers:
  - RAID-6 in each cluster node with auto-failover and a hot spare
  - Distributed filesystem on top
    - Replicates amongst the cluster nodes so that nodes can fail
  - And then distribute to a different WSC...