Lecture 38

Dependability and ECC

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

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)

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

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 as we saw with Rowhammer: Errors aren't just errors, but can be potential avenues for exploitation!

Dependability via Redundancy: Time vs. Space

- *Spatial Redundancy* – replicated data or check 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 hope rarely down, shorthand is "number of 9s of availability per year"
- 1 nine: 90% => 36 days of repair/year
- 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

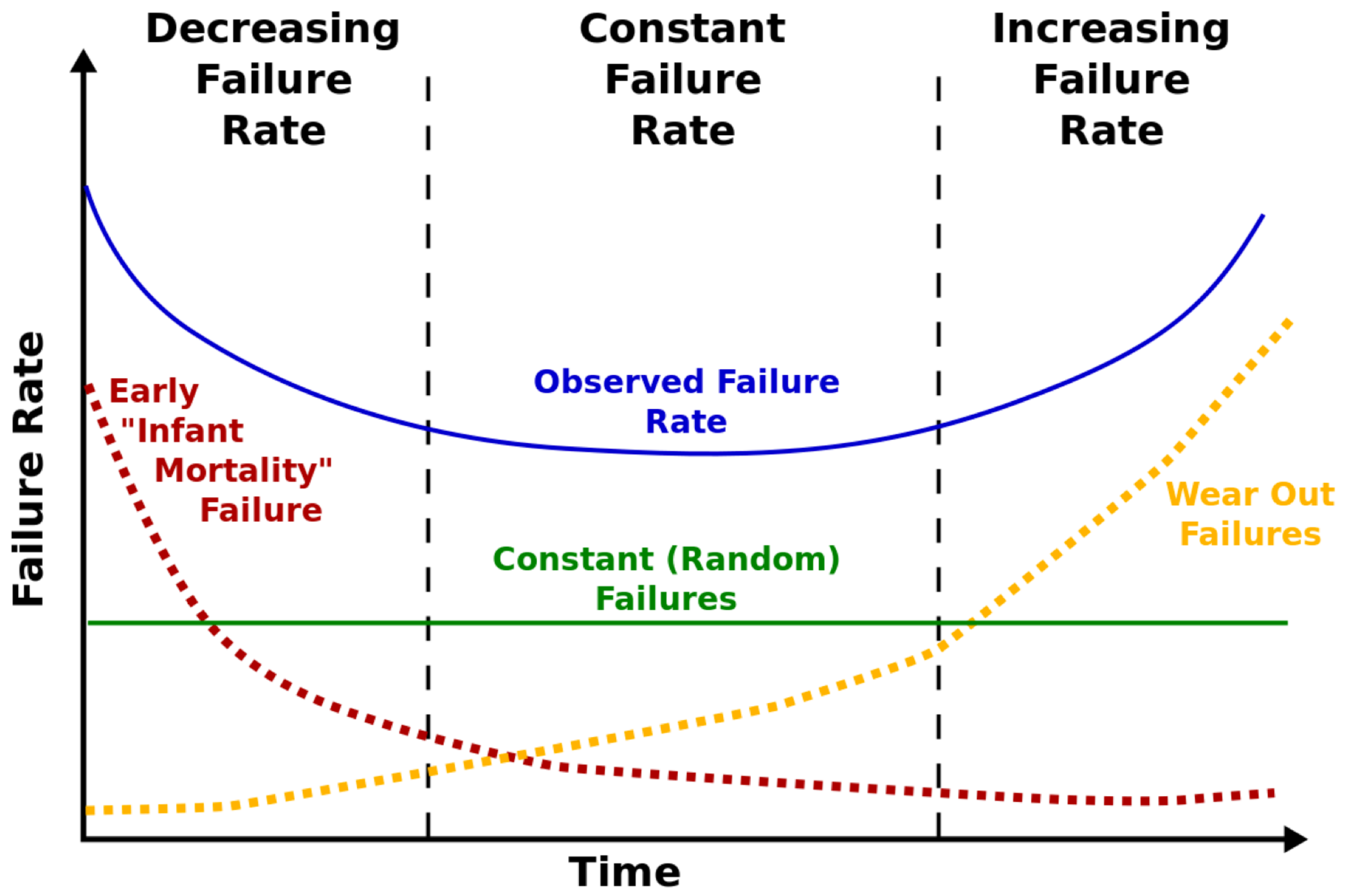
Reliability Measures

- 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
  - "Crib death"
- 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

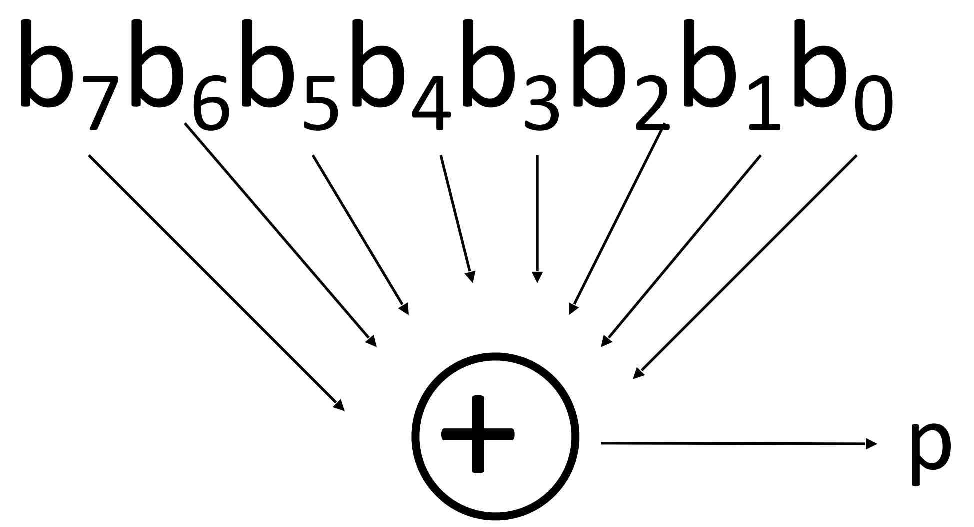
Block Code Principles

- 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?

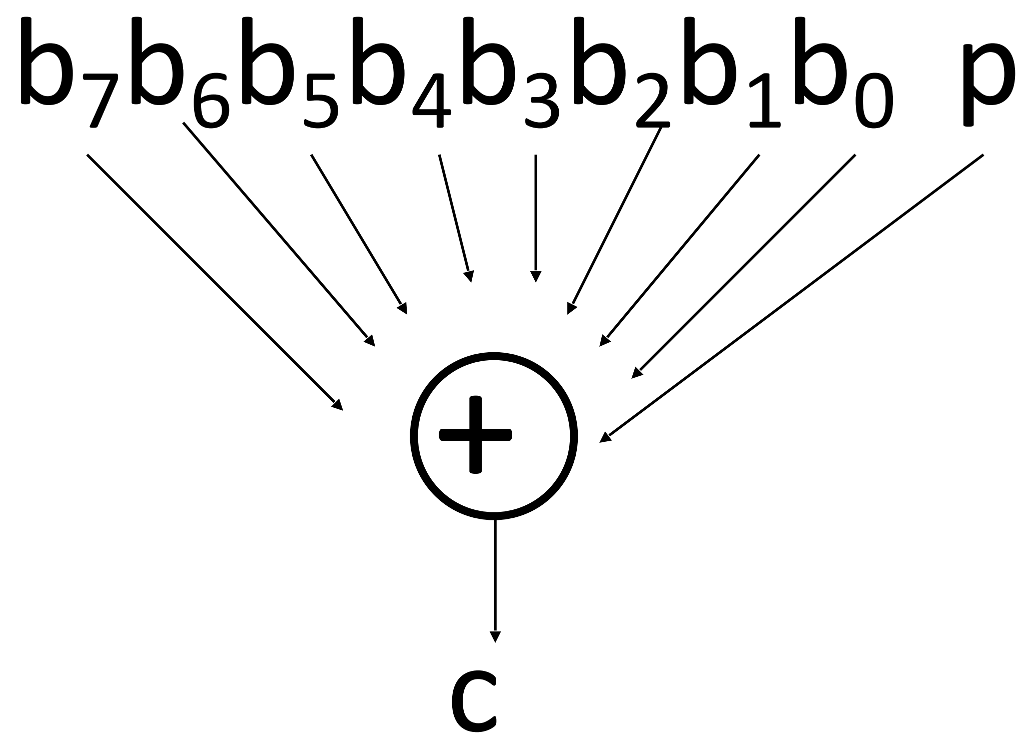
Richard Hamming, 1915-98
Turing Award Winner

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

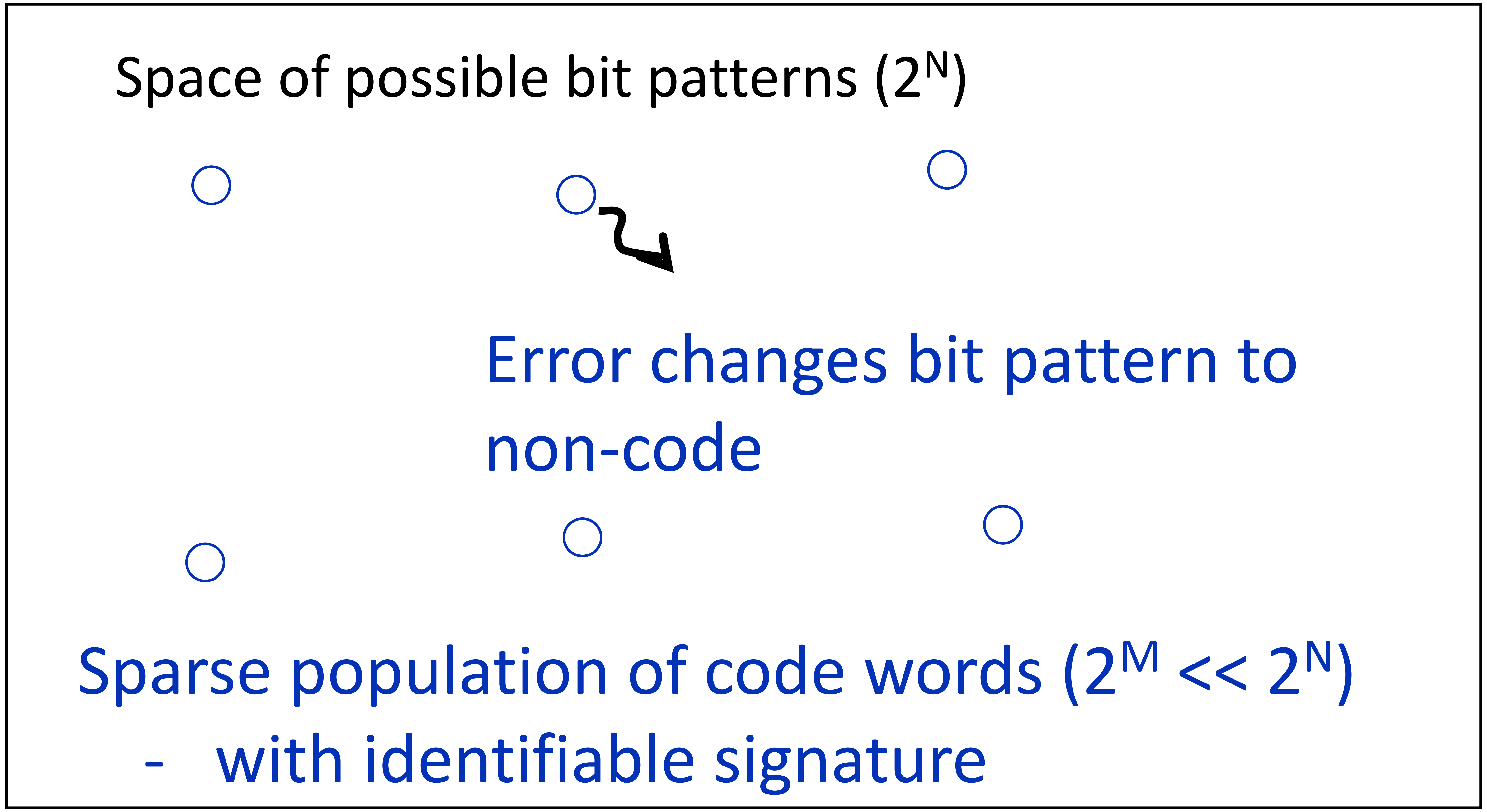
- Read from memory
  0101 0101 0
- 4 ones => even parity, so no error
- Read from memory
  1101 0101 0
- 5 ones => odd parity, so error
- What if error in parity bit?

Suppose Want to Correct 1 Error?

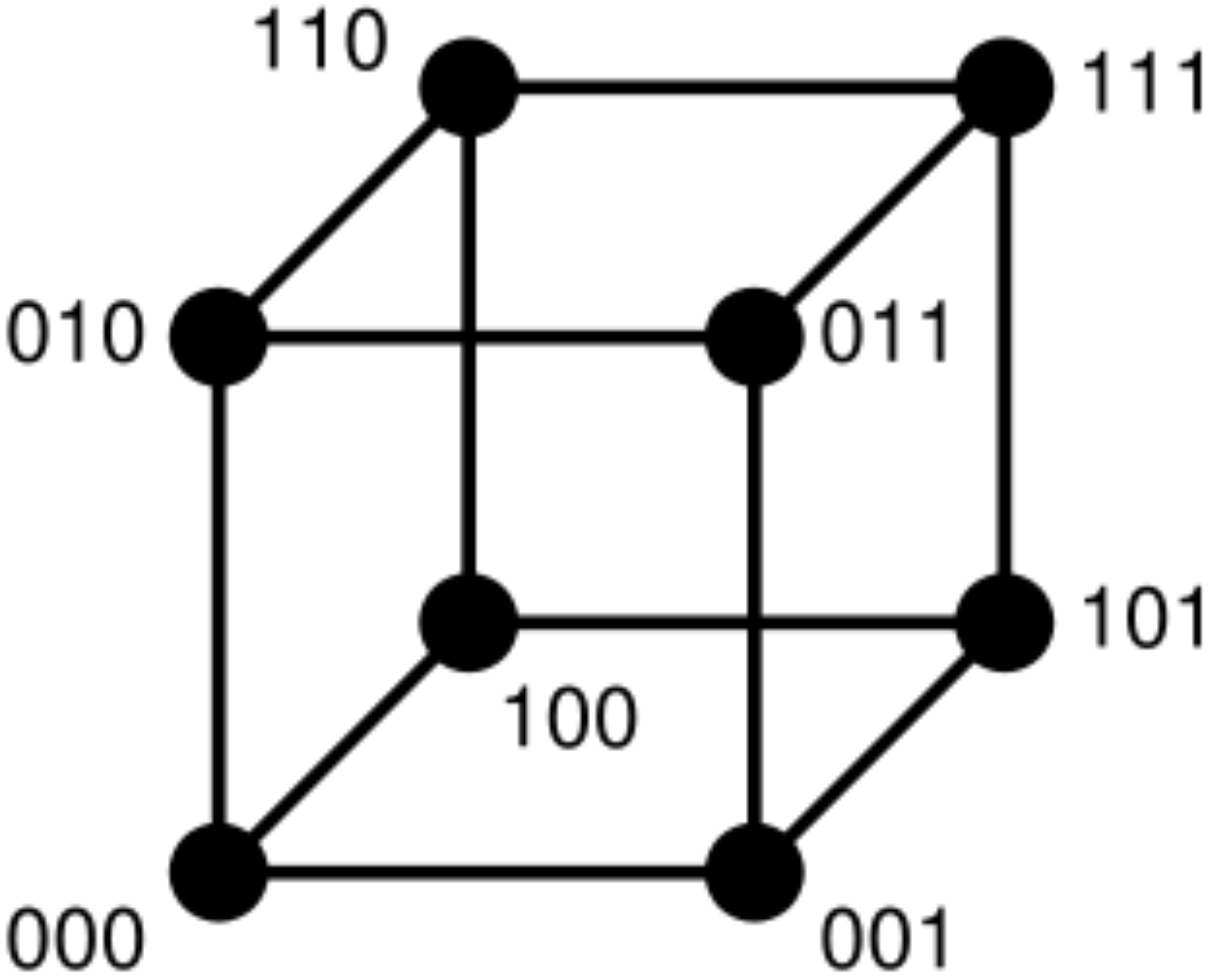
- Richard Hamming came up with simple to understand mapping to allow Error Correction at minimum distance of 3
  - Single error correction, double error detection
- Called "Hamming ECC"
  - Worked weekends on relay computer with unreliable card reader, frustrated with manual restarting
  - Got interested 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

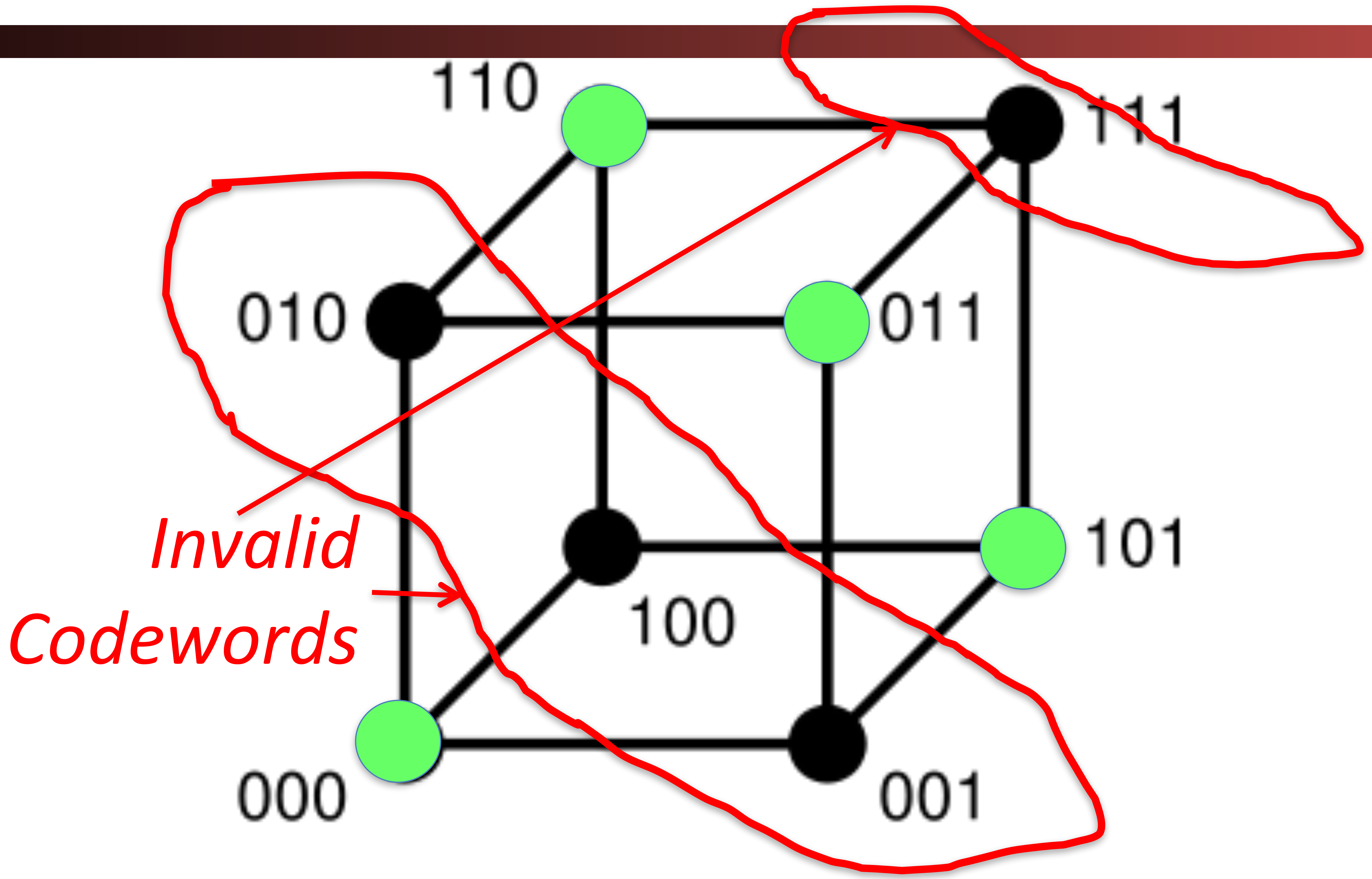


Hamming Distance: 8 code words



Hamming Distance 2: Detection

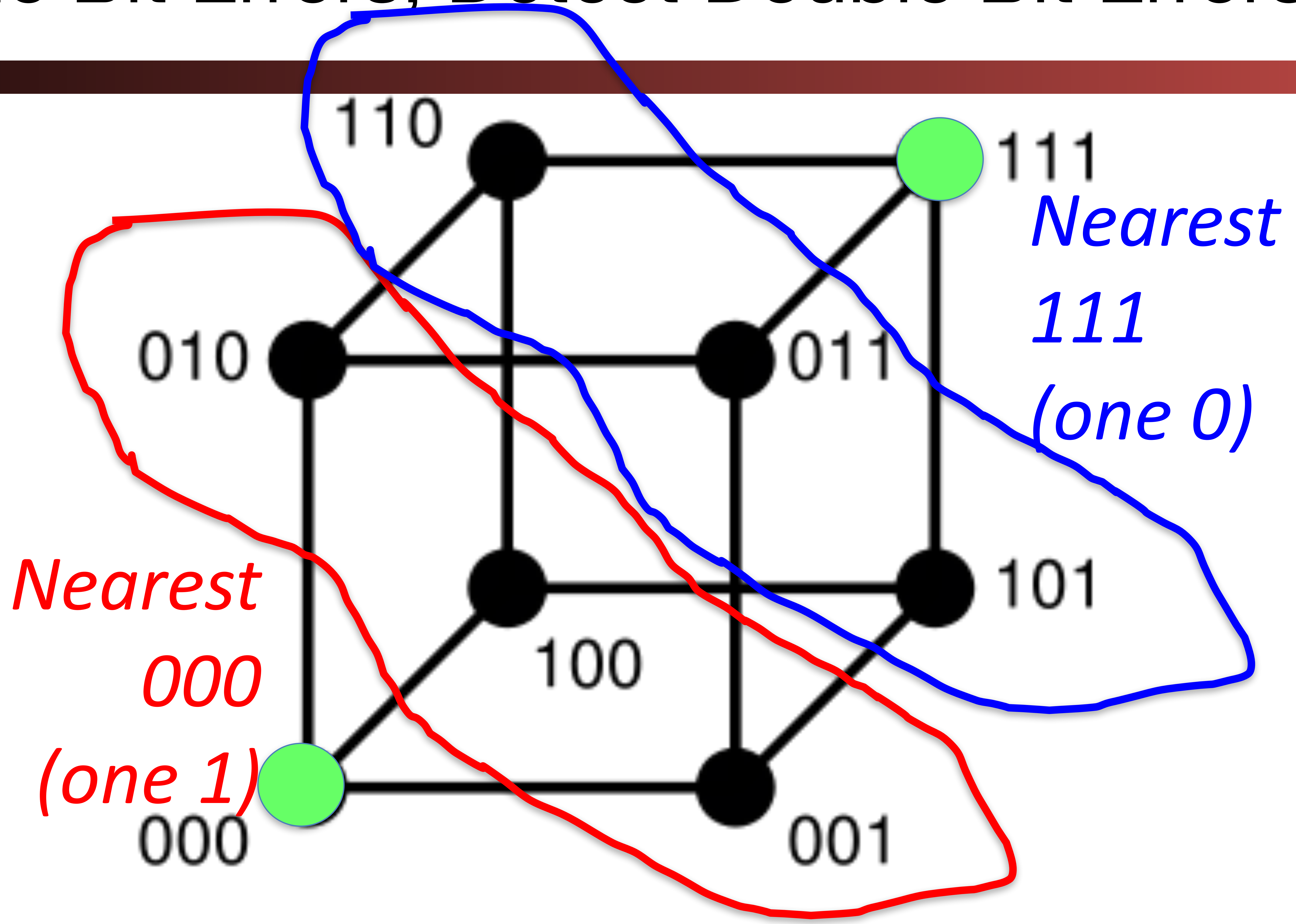
Detect Single Bit Errors



- 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

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

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 _:
- Position 2 checks bits **2,3,6,7,10,11**:
  0 **? 1** _ 0 **0 1** _ 1 **0 1** 0. set position 2 to a _:
- Position 4 checks bits **4,5,6,7,12**:
  0 1 1 **? 0 0 1** _ 1 0 1 **0**. set position 4 to a _:
- 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 _:

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 0:
  0 _ 1 _ 0 0 1 _ 1 0 1 0
- Position 2 checks bits 2,3,6,7,10,11:
  0 ? 1 _ 0 0 1 _ 1 0 1 0. set position 2 to a 1:
  0 1 1 _ 0 0 1 _ 1 0 1 0
- Position 4 checks bits 4,5,6,7,12:
  0 1 1 ? 0 0 1 _ 1 0 1 0. set position 4 to a 1:
  0 1 1 1 0 0 1 _ 1 0 1 0
- 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 0:
  0 1 1 1 0 0 1 0 1 0 1 0

Hamming ECC

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

Hamming ECC Error Check

- Suppose receive

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

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

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 …

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

Hamming ECC Error Correct

- Suppose receive

```
011100101010
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
  - 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"

And, in Conclusion, …

- Great Idea: Redundancy to Get Dependability
  - Spatial (extra hardware) and Temporal (retry if error)
- Reliability: MTTF & Annualized Failure Rate (AFR)
- Availability: % uptime (MTTF-MTTR/MTTF)
- Memory
  - Hamming distance 2: Parity for Single Error Detect
  - Hamming distance 3: Single Error Correction Code + encode bit position of error
- Treat disks like memory, except you know when a disk has failed—erasure makes parity an Error Correcting Code
- RAID-2, -3, -4, -5: Interleaved data and parity