CS 61C: Great Ideas in Computer Architecture

Dependability

Guest Lecturer:

Paul Ruan

Review

- WSC
  - Servers on a rack, rack part of cluster
  - Issues to handle include load balancing, failures, power usage
  - PUE = Total building power / IT equipment power
- Request Level Parallelism
- MapReduce Data Level Parallelism
  - Framework to divide up data to be processed in parallel
  - Mapper outputs intermediate kv-pairs
  - Reducer “combines” intermediate values with same key

Agenda

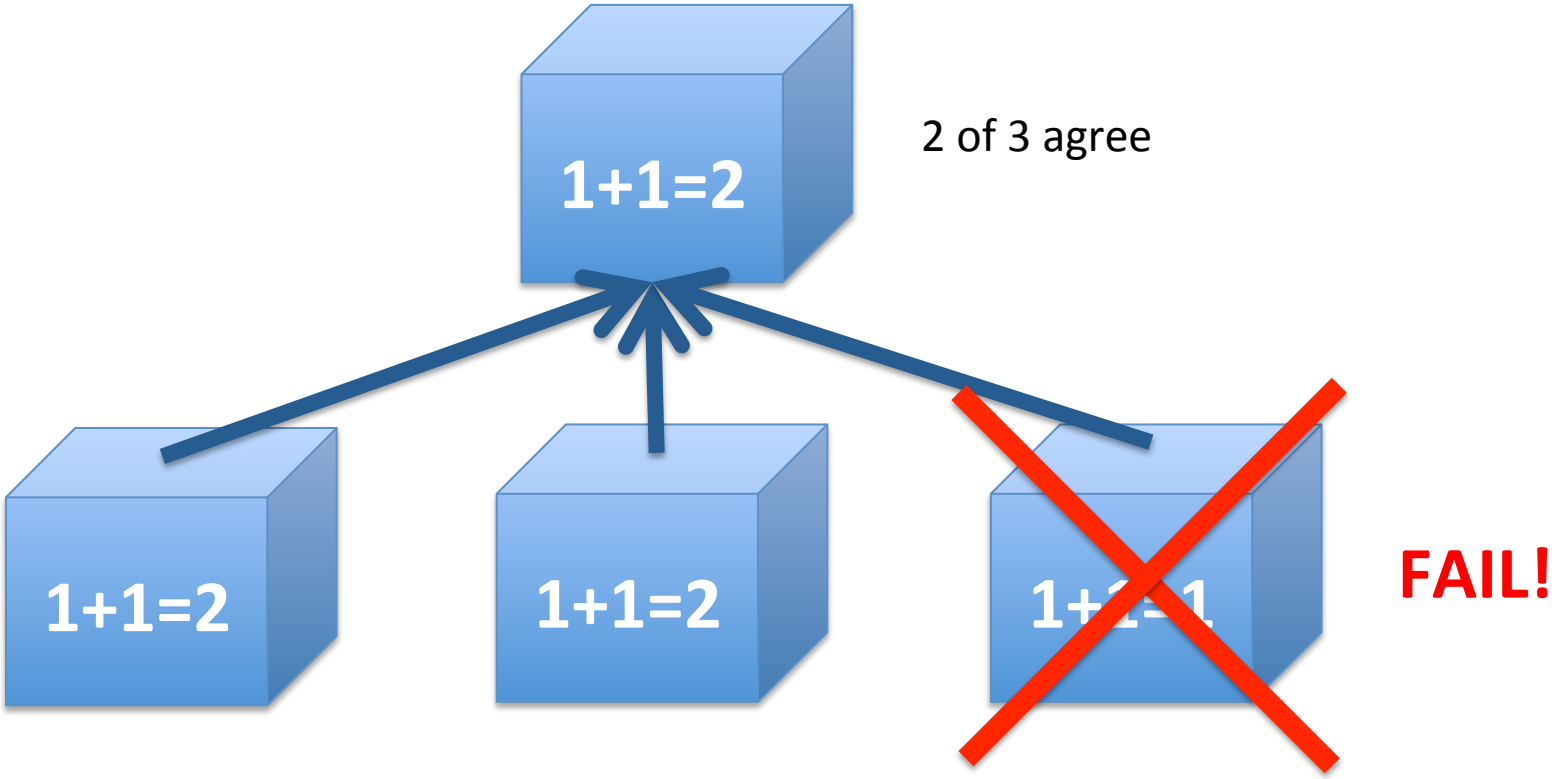
- Dependability
- Administrivia
- RAID
- Error Correcting Codes

Review - 6 Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. **Dependability via Redundancy**

Great Idea #6: Dependability via Redundancy

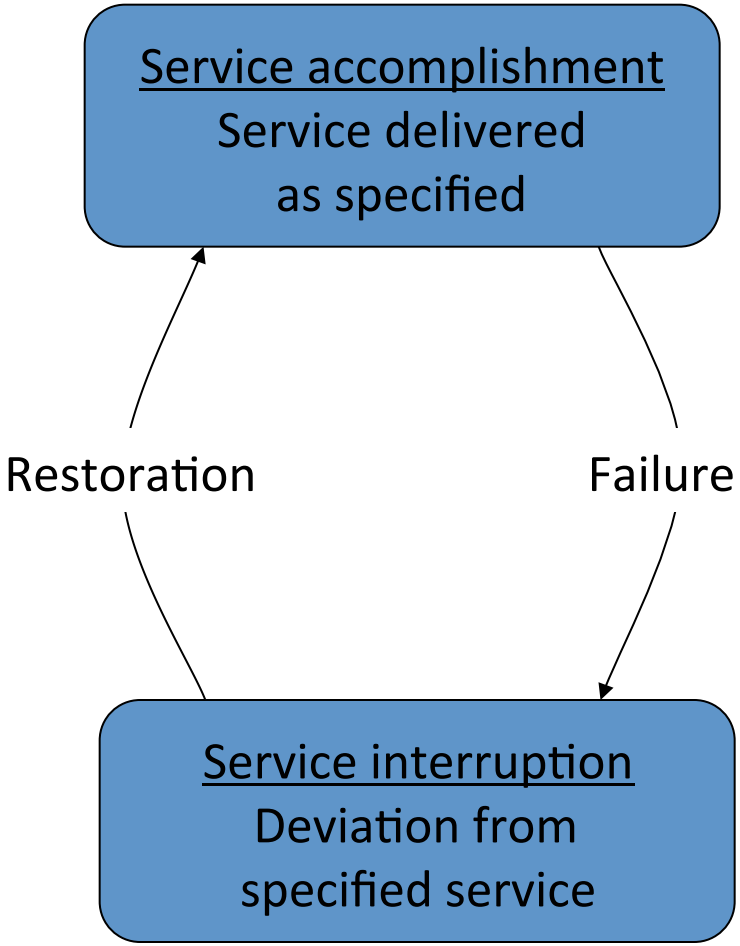
- Redundancy so that a failing piece doesn't make the whole system fail



Great Idea #6: Dependability via Redundancy

- Applies to everything from datacenters to memory
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant routes so can lose nodes but Internet doesn't fail
  - 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



- Fault: failure of a component
  - May or may not lead to system failure

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) = MTTF / MTBF
- Improving Availability
  - Increase MTTF: More reliable hardware/software + Fault Tolerance
  - Reduce MTTR: improved tools and processes for diagnosis and repair

Reliability Measures

- MTTF, MTBF usually measured in hours
  - E.g., average MTTF is 100,000 hours
- Another measure is average number of failures per year
  - E.g., 1000 disks with 100,000 hour MTTF
  - 365 days * 24 hours = 8760 hours
  - (1000 disks * 8760 hrs/year) / 100,000 hrs = 87.6 failed disks per year on average
  - 87.6/1000 = 8.76% annualized failure rate (AFR)

Availability Measures

- Availability = MTTF / (MTTF + MTTR) as %
- 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

Dependability Design Principle

- No single points of failure
- "Chain is only as strong as its weakest link"
- Dependability Corollary of Amdahl's Law
  - Doesn't matter how dependable you make one portion of system
  - Dependability limited by part you do not improve

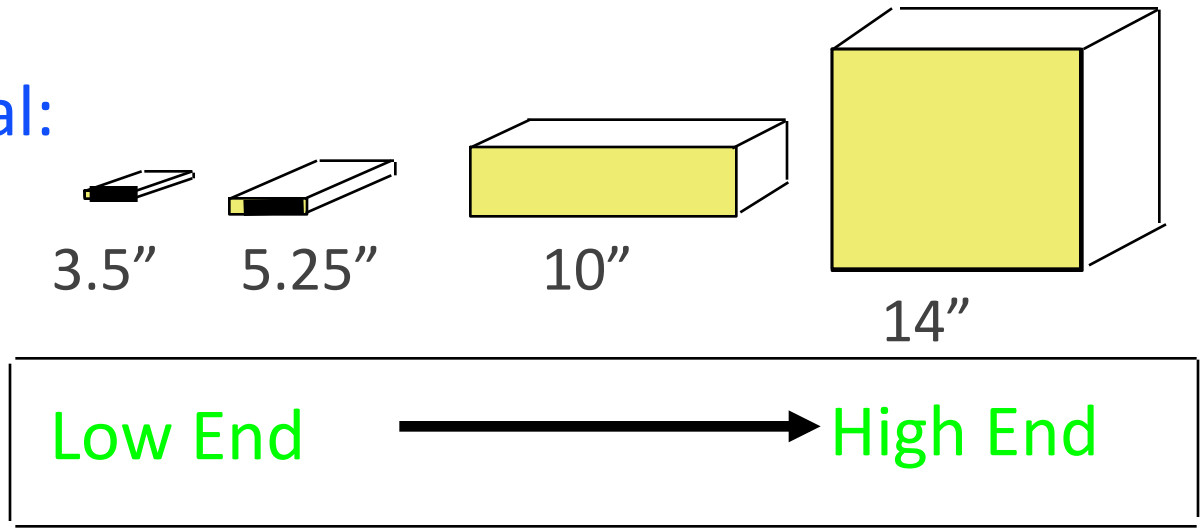
Agenda

- Dependability
- RAID
- Administrivia
- Error Correcting Codes

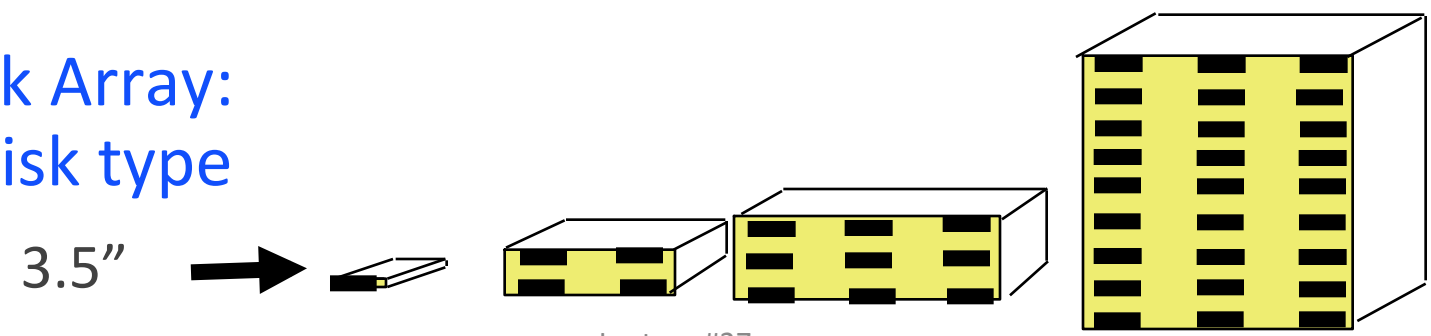
Arrays of Small Disks

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

Conventional:
4 disk types



Disk Array:
1 disk type



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	120 MB/s	8X
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?

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

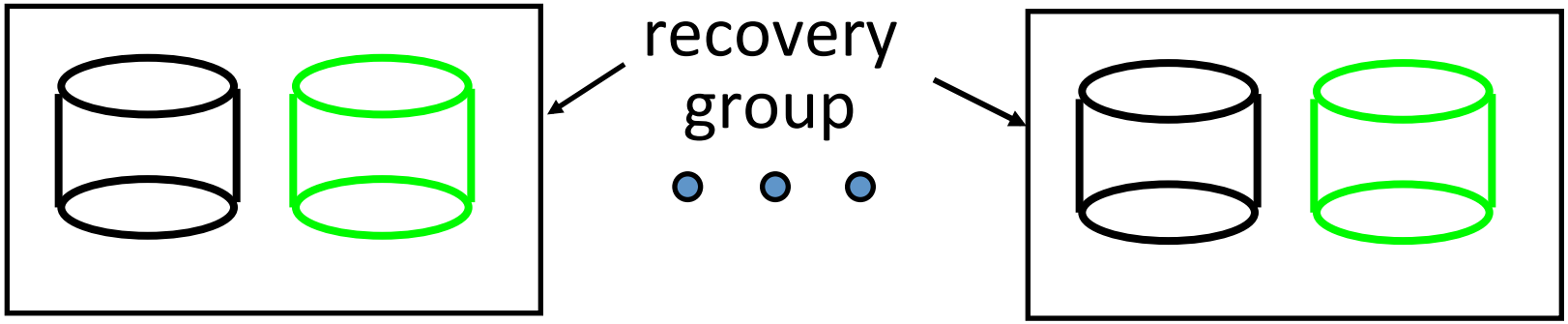
	IBM 3390K	IBM 3.5" 0061	x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	120 MB/s	8X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	~700 Hrs	
Cost	$250K	$2K	$150K	

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

RAID: Redundant Arrays of Inexpensive Disks

- Files are "striped" across multiple disks
- Redundancy yields high data 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

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



- Each disk is fully duplicated onto its "mirror"
  - Very high availability can be achieved
- Bandwidth sacrifice on write:
  - Logical write = two physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead

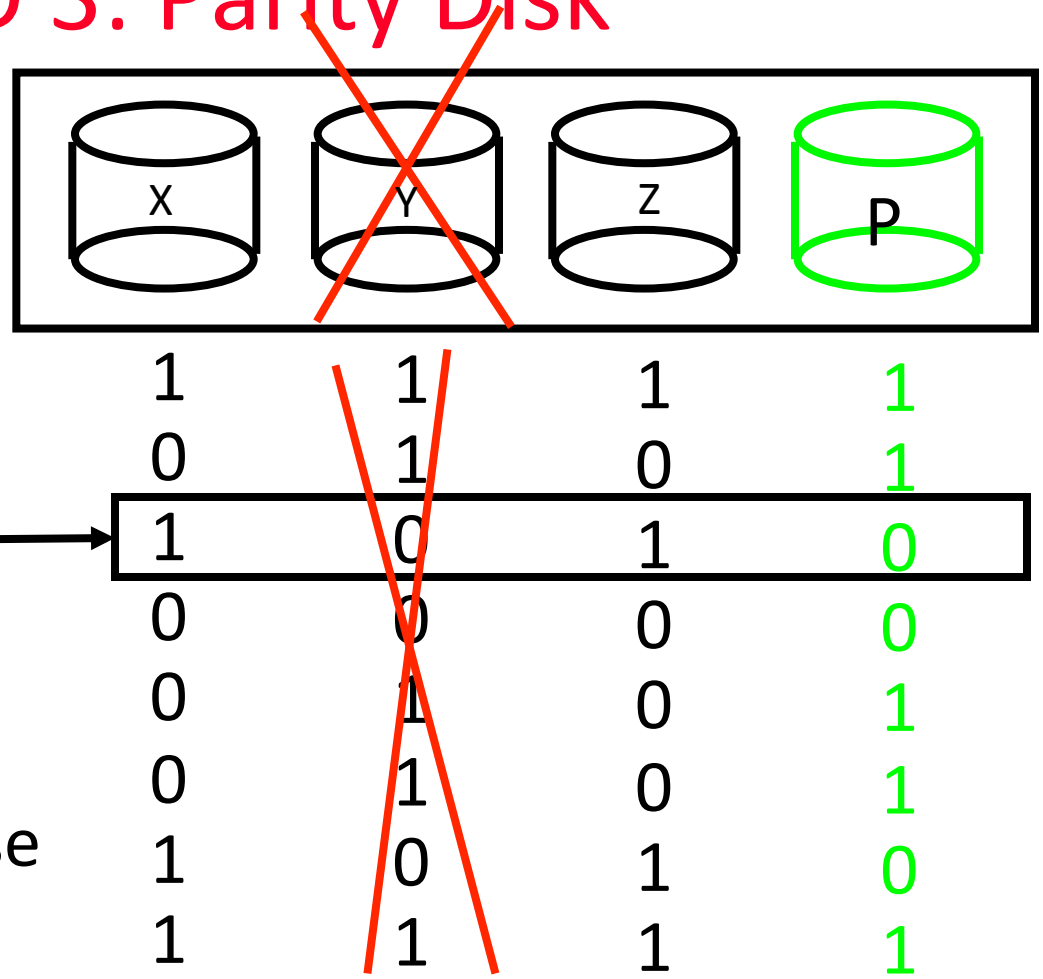
Parity Bit

- Describes whether group of bits contains an even or odd number of 1's
- Assume from now that 1 means odd and 0 means even
  - Can use xor to compute parity bit
- Adding the parity bit to group will always result in an even number of 1's ("even parity")
  - 100 Parity: 1
  - 101 Parity: 0
- If we know number of 1's must be even, can we figure out what a single missing bit should be?
  - 10?11

Redundant Array of Inexpensive Disks
RAID 3: Parity Disk

10010011
11001101
10010011
. . .

logical record
Striped physical records

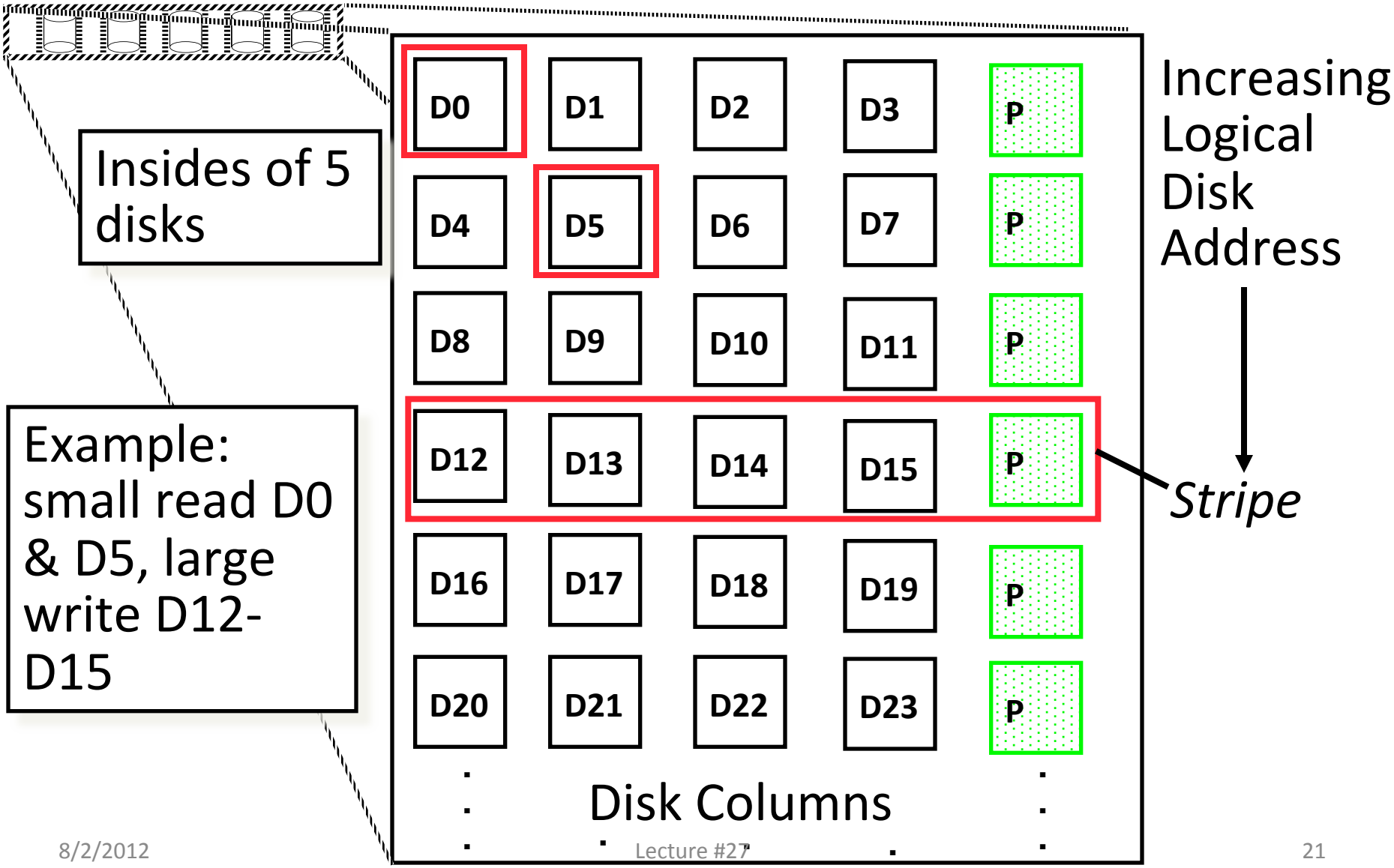


P contains parity bits of the other disks.
If any disk fails, can use other disks to recover data.

Administrivia

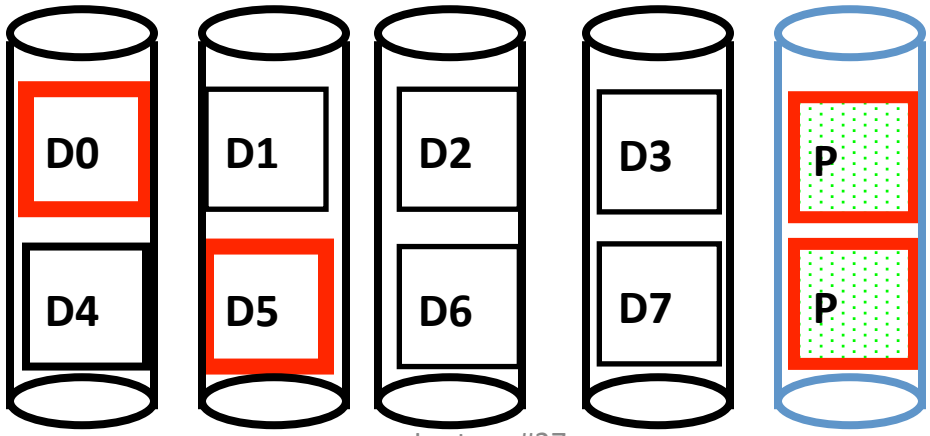
- Project 3 (individual) due Sunday 8/5
- Final Review – Friday 8/3, 3-6pm in 306 Soda
- Final – Thurs 8/9, 9am-12pm, 245 Li Ka Shing – Focus on 2nd half material, though midterm material still fair game
  - MIPS Green Sheet provided again
  - Two-sided handwritten cheat sheet
    - Can use the back side of your midterm cheat sheet!

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



Inspiration for RAID 5

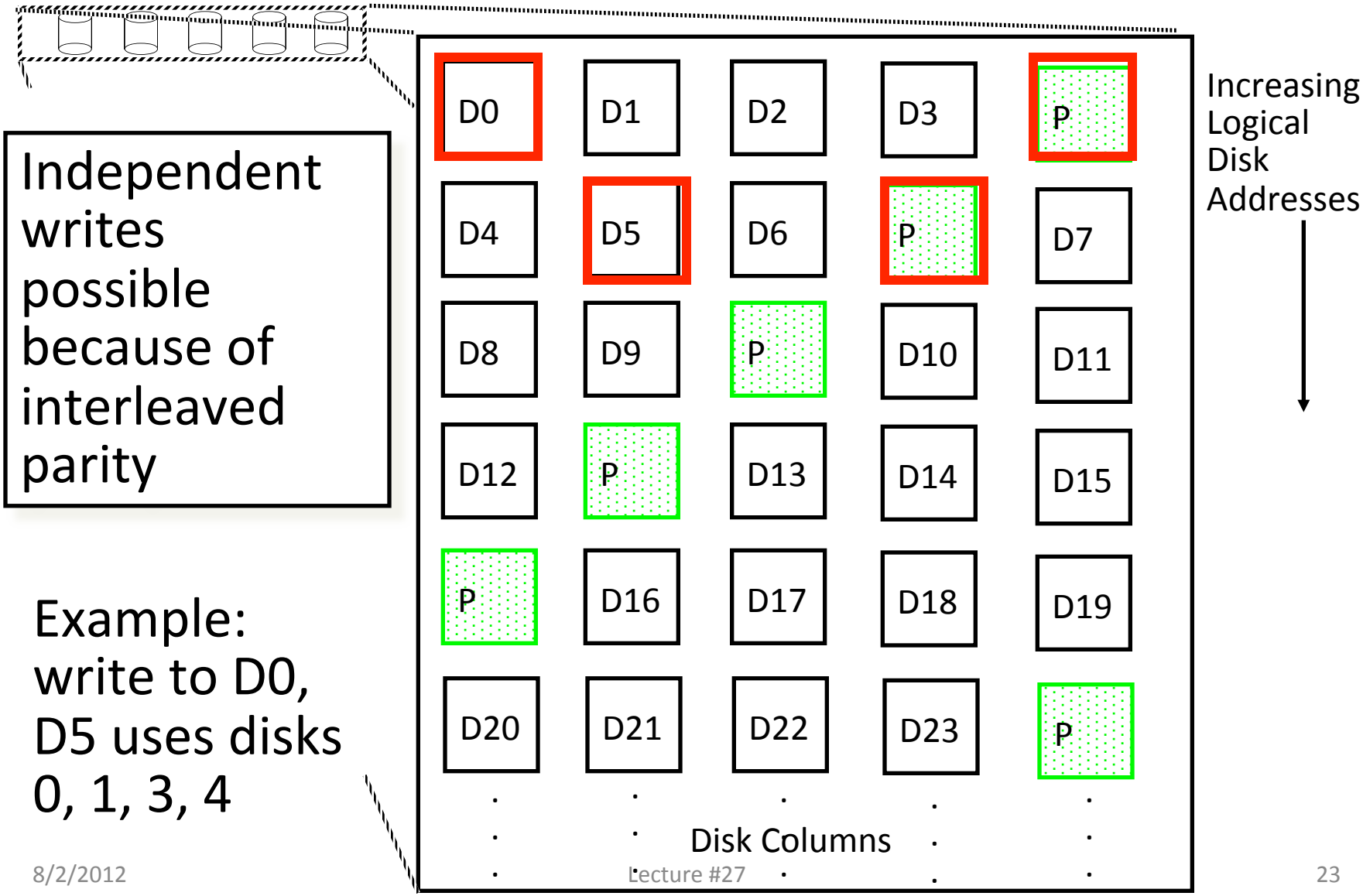
- When writing to a disk, need to update Parity
- Small writes are bottlenecked by Parity Disk: Write to D0, D5 both also write to P disk



RAID 5: High I/O Rate Interleaved Parity

Independent writes possible because of interleaved parity

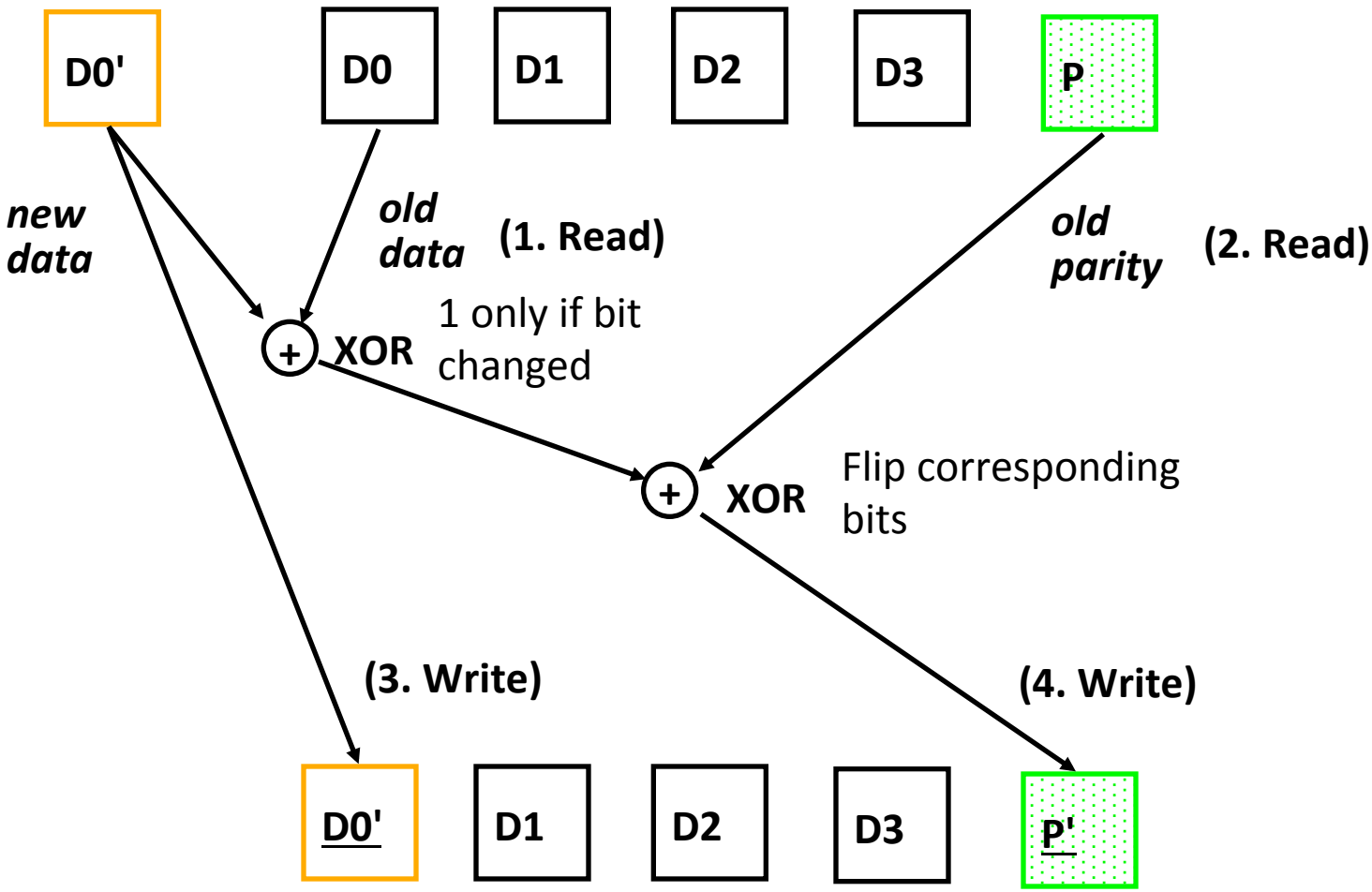
Example: write to D0, D5 uses disks 0, 1, 3, 4



Updating the Parity Data

RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



RAID 6: Recovering from 2 failures

- Why > 1 failure recovery?
  - operator accidentally replaces wrong disk during a failure
  - since disk bandwidth is growing more slowly than disk capacity, the MTT Repair a disk in a RAID system is increasing
    ⇒increases the chances of a 2nd failure during repair since takes longer
  - reading much more data during reconstruction meant increasing the chance of an uncorrectable media failure during read, which would result in data loss

Agenda

- Dependability
- Administrivia
- RAID
- Error Correcting Codes

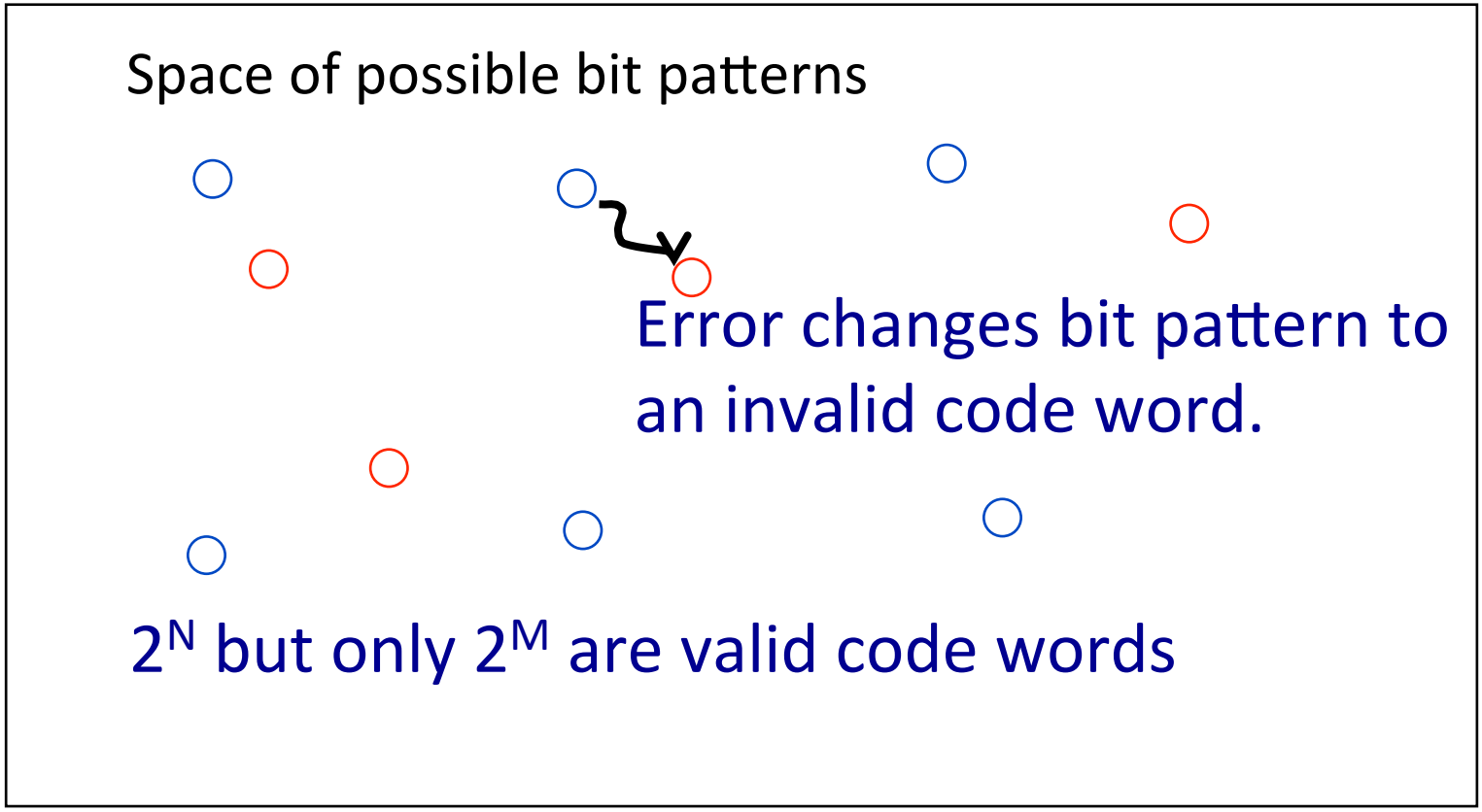
Error Detection/Correction Codes

- 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 occur when chips permanently fail.
  - Problem gets worse as memories get denser and larger

Error Detection/Correction Codes

- Protect against errors with EDC/ECC
- Extra bits are added to each M-bit data word to produce an N-bit code word.
  - Extra bits are a function of the data
  - Each data word value is mapped to a valid code word.
  - Certain errors change valid code words to invalid ones.

Detecting/Correcting Code Concept



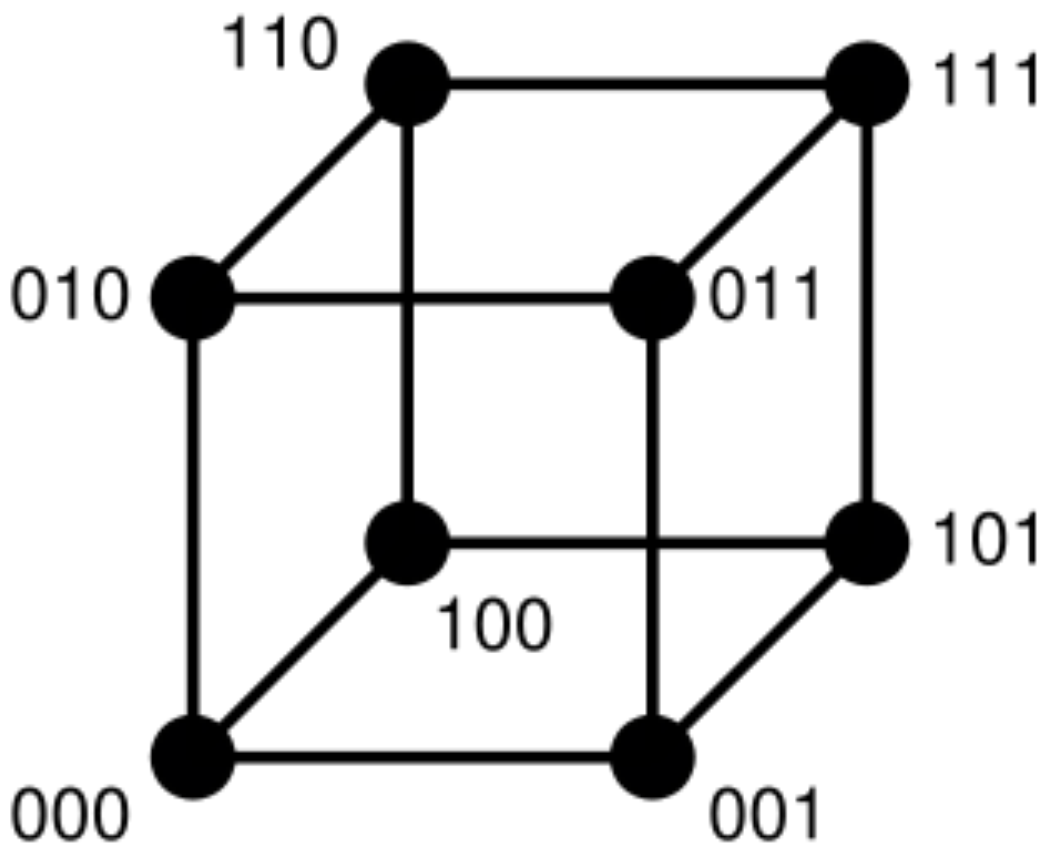
- Detection: fails code word validity check
- Correction: map to nearest valid code word

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) = ?
- Consider if the minimum distance between valid code words is 2 and get a 1 bit error.
  - Can we detect it?
  - Can we correct it?

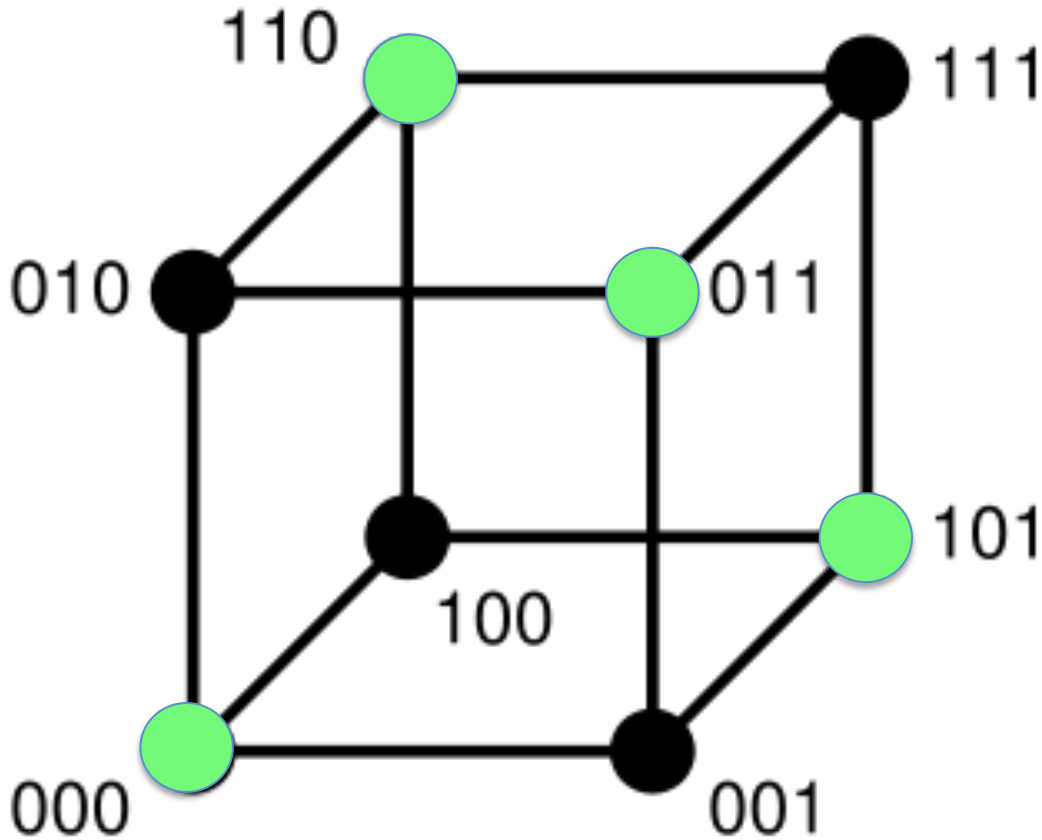
Richard Hamming, 1915-98
Turing Award Winner

3 Bit Example



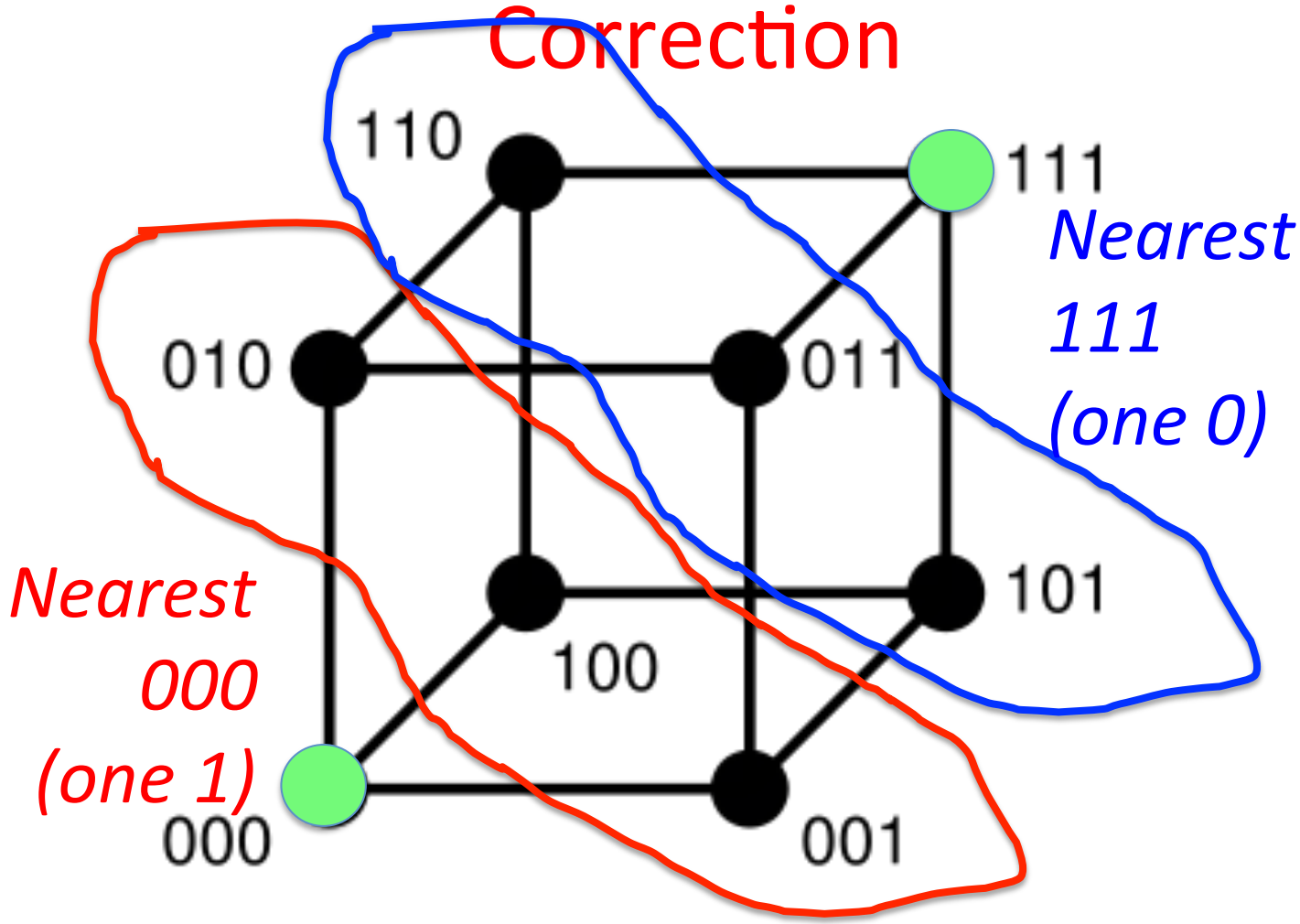
Organized this way, each edge represents a Hamming distance of 1.

Minimum Hamming Distance 2: Detection



- No 1 bit error goes to another valid code word
- ½ of the code words are valid

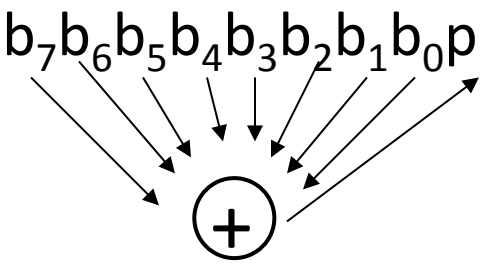
Minimum Hamming Distance 3: Correction



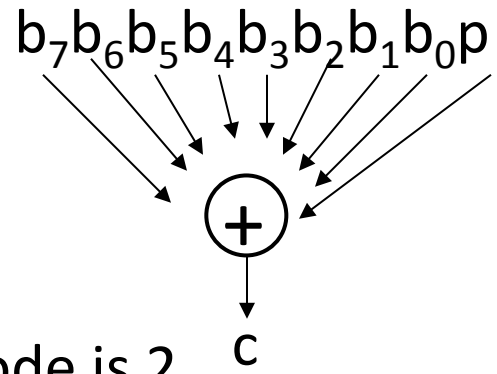
- No 2 bit error goes to another valid code
- 1/4 of the code words are valid

Parity: Simple Error Detection Coding

- Add parity bit when writing block of data



- When data is read, check parity:
  - Ok if even number of 1s
  - Error if otherwise



- Minimum Hamming distance of parity code is 2
- A non-zero parity indicates an error occurred:
  - Two bit errors are not detected (nor any even number of errors)
  - 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?

- Can we correct if minimum distance is 2?
- What must minimum distance be?
- Richard Hamming came up with simple to understand mapping to allow Error Correction at minimum distance of 3
- Called Hamming ECC for Error Correction Code

Get To Know Your Staff

- Category: **Wishlist**

	Paul	**Sung Roa**
Bucketlist Item	Build a giant sandcastle	Sky diving
Skill	Drawing	Singing
Language	French	Russian
Superpower	Time Travel	Teleportation

Hamming Error Correction Code

- Use of extra parity bits to allow the position identification of a single error

1. Mark all bit positions in the code word that are powers of 2 as parity bits. (positions 1, 2, 4, 8, 16, …)
   - Start numbering bits at 1 at left, not at 0 on right
2. All other bit positions are for the data bits. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, …)

Hamming ECC

3. Each parity bit calculates the parity for some of the bits in the code word

- The position of parity bit determines group of bits that it checks
- Bit 1 (0001_2): checks bits 1,3,5,7, … ($XXX1_2$)
- Bit 2 (0010_2): checks bits 2,3,6,7, … ($XX1X_2$)
- Bit 4 (0100_2): checks bits 4-7, 12-15, … ($X1XX_2$)
- Bit 8 (1000_2): checks bits 8-15, 24-31, … ($1XXX_2$)

Hamming ECC

4. Set parity bits to create even parity for each group

- A byte of data: 10011010
- Create the code word, leaving spaces for the parity bits:
- $\__{1}\ \__{2}\ 1_{3}\ \__{4}\ 0_{5}\ 0_{6}\ 1_{7}\ \__{8}\ 1_{9}\ 0_{10}\ 1_{11}\ 0_{12}$
- Calculate the parity bits

Hamming ECC

- Position 1 checks bits 1,3,5,7,9,11:
  ? _ **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
- Position 4 checks bits 4,5,6,7,12:
  0 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**

Hamming ECC

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

Hamming ECC

- **Finding and fixing a corrupted bit:**
- Suppose receive $0_1 1_2 1_3 1_4 0_5 0_6 1_7 0_8 1_9 1_{10} 1_{11} 0_{12}$
- Parity bits 2 and 8 incorrect. As 2 + 8 = 10, bit position 10 is location of bad bit: flip value!
- Corrected value: 011100101010
- Why does Hamming ECC work?

Hamming Error Correcting Code

- Overhead involved in single error correction code
- Let p be total number of parity bits and d number of data bits in $p + d$ bit word
- If p error correction bits are to point to error bit ($p + d$ cases) + indicate that no error exists (1 case), we need:

  $2^p >= p + d + 1,$

  thus $p >= \log(p + d + 1)$

  for large d, p approaches $\log(d)$
- *8 bits data => $d = 8$, $2^p = p + 8 + 1 \Rightarrow p = 4$*
- *16 data => 5 parity, 32 data => 6 parity, 64 data => 7 parity*

Hamming Single Error Correction, Double Error Detection (SEC/DED)

- Adding extra parity bit covering the entire SEC code word provides double error detection as well

  1 2 3 4 5 6 7 8

  p_1 p_2 d_1 p_3 d_2 d_3 d_4 p_4

- Let H be the position of the incorrect bit we would find from checking p_1,p_2, and p_3 (note that 0 means no error) and let P be parity of complete code word.

  H=0 P=0, no error

  H≠0 P=1, correctable single error (odd parity if 1 error => p_4=1)

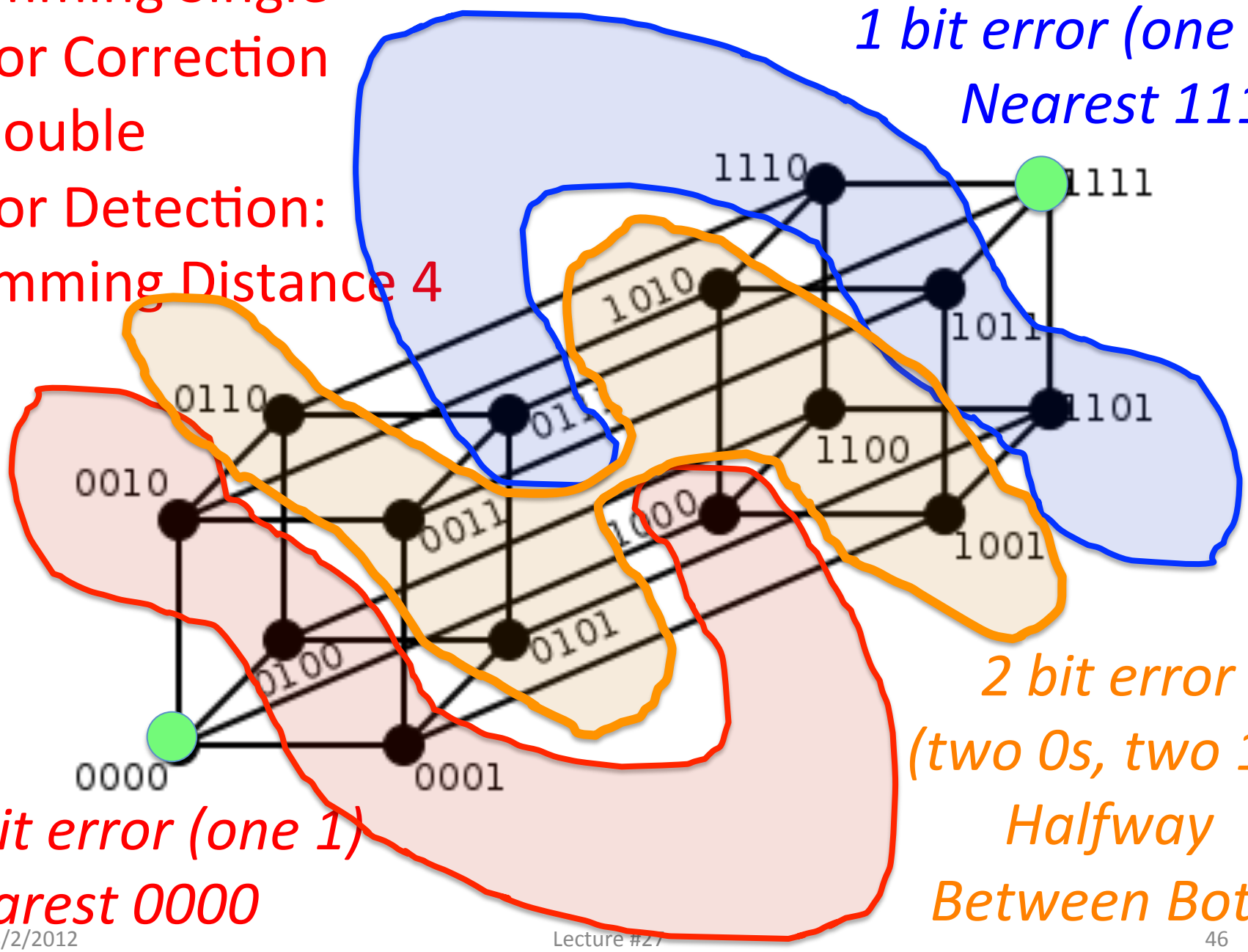
  H≠0 P=0, double error occurred (even parity if 2 errors=> p_4=0)

  H=0 P=1, an error occurred in p_4 bit, not in rest of word

Typical modern codes in DRAM memory systems:

64-bit data blocks (8 bytes) with 72-bit code words (9 bytes).

Hamming Single Error Correction + Double Error Detection: Hamming Distance 4



What if More Than 2 Bit Errors?

- Network transmissions, disks, distributed storage common failure mode is bursts of bit errors, not just one or two bit errors
  - contiguous sequence of *B* bits in which first, last and any number of intermediate bits are in error
  - caused by impulse noise or by fading in wireless
  - effect is greater at higher data rates
- Other tools: cyclic redundancy check, Reed-Solomon, other linear codes

Summary

- Great Idea: Redundancy to Get Dependability
- Reliability: MTTF & Annual failure rate
- Availability: % uptime (MTTF-MTTR/MTTF)
- RAID: Redundant Arrays of Inexpensive Disks
  - Improve I/O rate while ensuring dependability
- Memory Errors:
  - Hamming distance 2: Parity for Single Error Detect
  - Hamming distance 3: Single Error Correction Code + encode bit position of error
  - Hamming distance 4: SEC/Double Error Detection