CS 110
Computer Architecture

Dependability and RAID

Instructor:

Sören Schwertfeger and Chundong Wang

https://robotics.shanghaitech.edu.cn/courses/ca/21s

School of Information Science and Technology

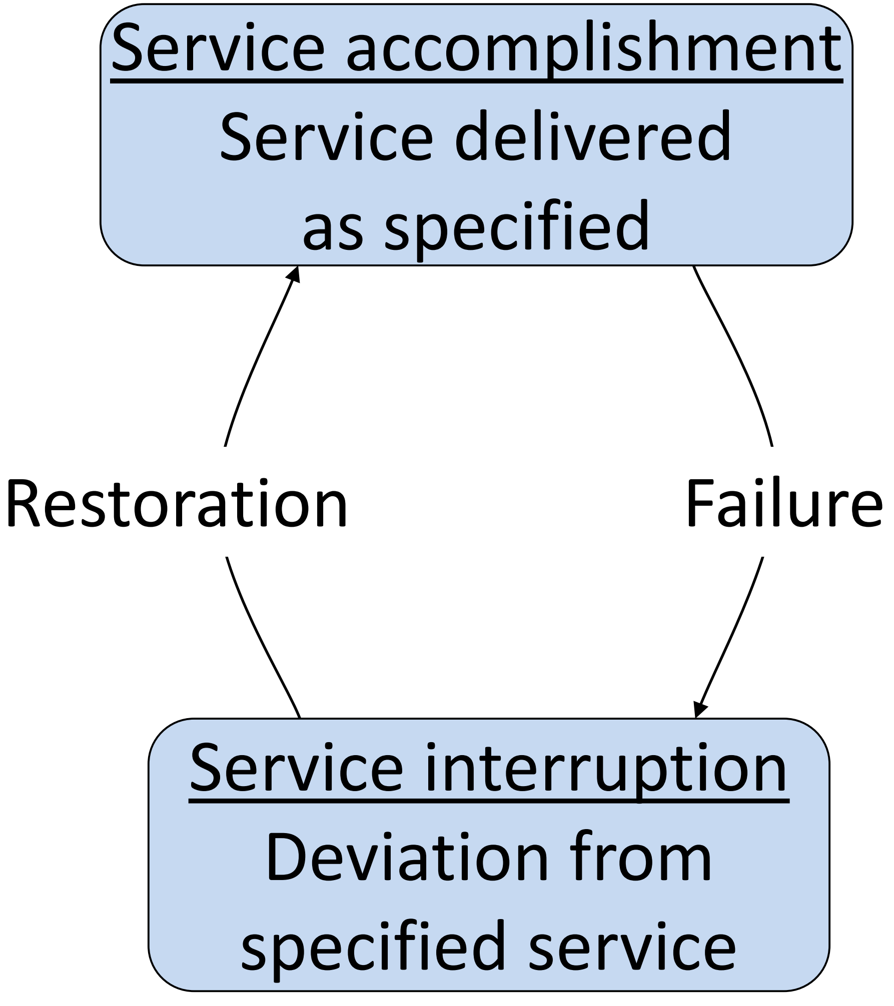
ShanghaiTech University

Slides based on UC Berkley's CS61C

Review

- I/O gives computers their 5 senses
- I/O speed range is 100-million to one
- Polling vs. Interrupts
- DMA to avoid wasting CPU time on data transfers
- Disks and flash for persistent storage
- Networking
  - Connecting computers, and networks
- Advanced Caches
  - LRU/MRU, inclusive/exclusive/non-inclusive
  - LLC slices, Scratchpad memory, etc.

Dependability



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

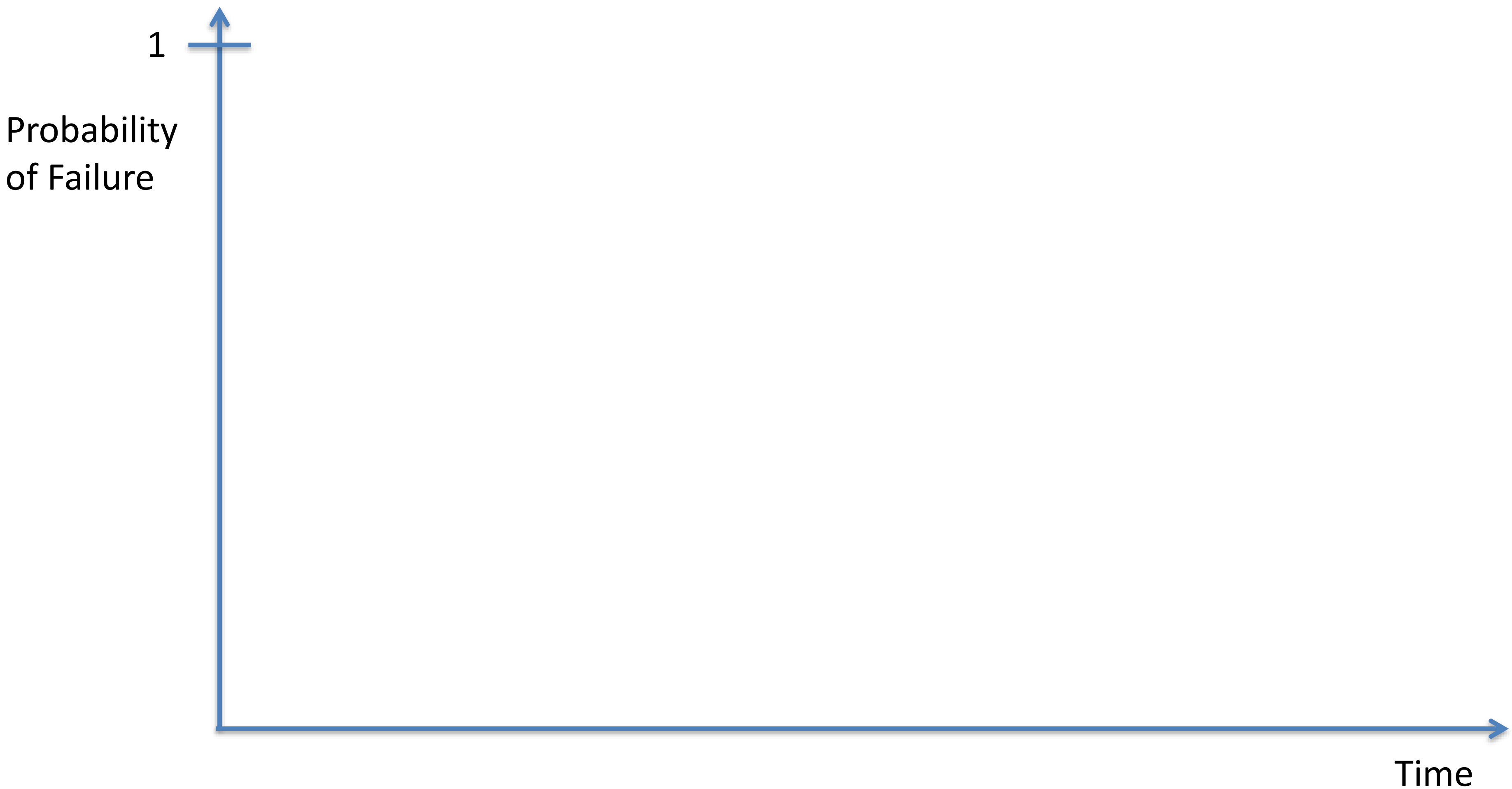
Dependability via Redundancy: Time vs. Space

- *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

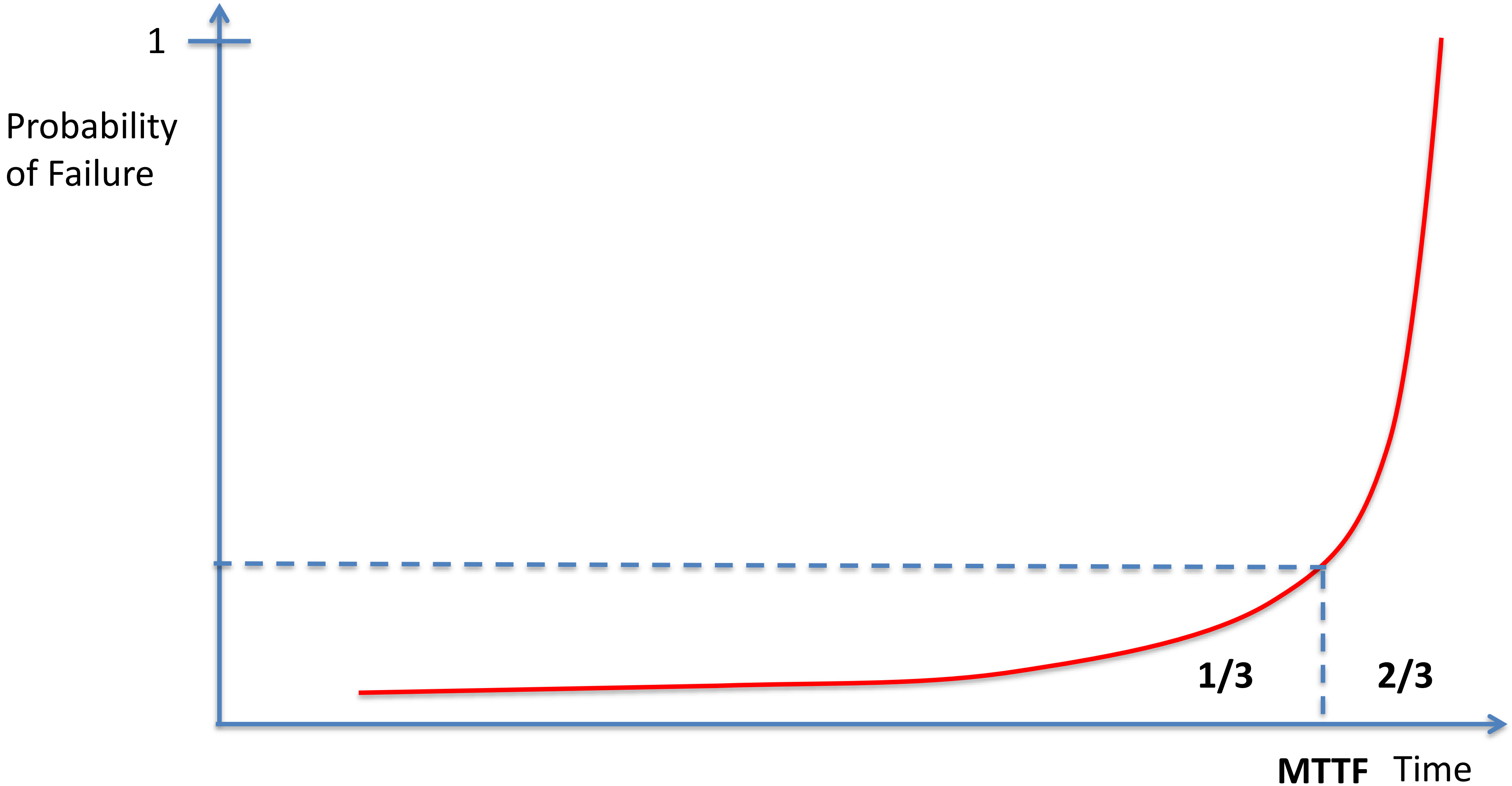
Dependability Measures

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

Understanding MTTF



Understanding MTTF



Availability Measures

- Availability = $\frac{MTTF}{MTTF+MTTR}$ 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 hours 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*

Dependability Design Principle

- Design Principle: No single points of failure
  - "Chain is only as strong as its weakest link"
  - *Achilles' Heel*
- 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

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 = 0<u>1</u>1<u>0</u>11, q = 0<u>0</u>1<u>1</u>11, Ham. distance (p,q) = 2
- p = 011011,
  q = 110001,
  distance (p,q) = ?
- Can think of extra bits as creating a code with the data
  - There is Ham. distance between codes

Richard Hamming, 1915-98
Turing Award Winner

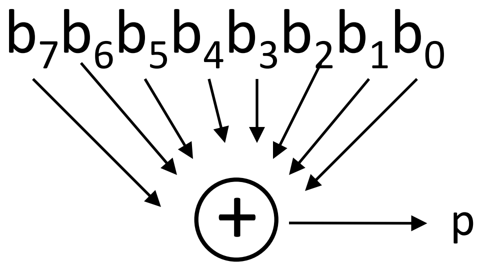
Parity

- Parity bits are added to a word to make it
  – either odd: odd numbers of ‘1’
  – or even: even number of ‘1’
- Let us add one parity bit to three-bit word

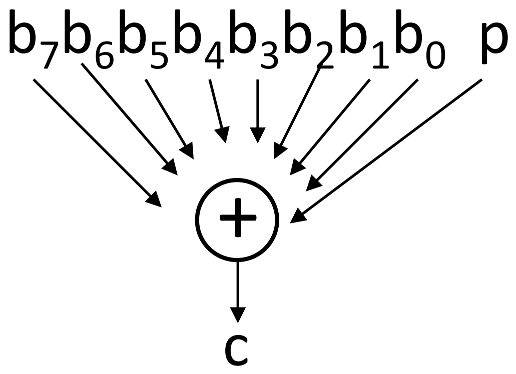
Odd Parity		Even Parity	
000	0001	000	0000
100	1000	100	1001
101	1011	101	1010
111	1110	111	1111

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).



- 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
- Minimum Hamming distance of valid parity codes is 2

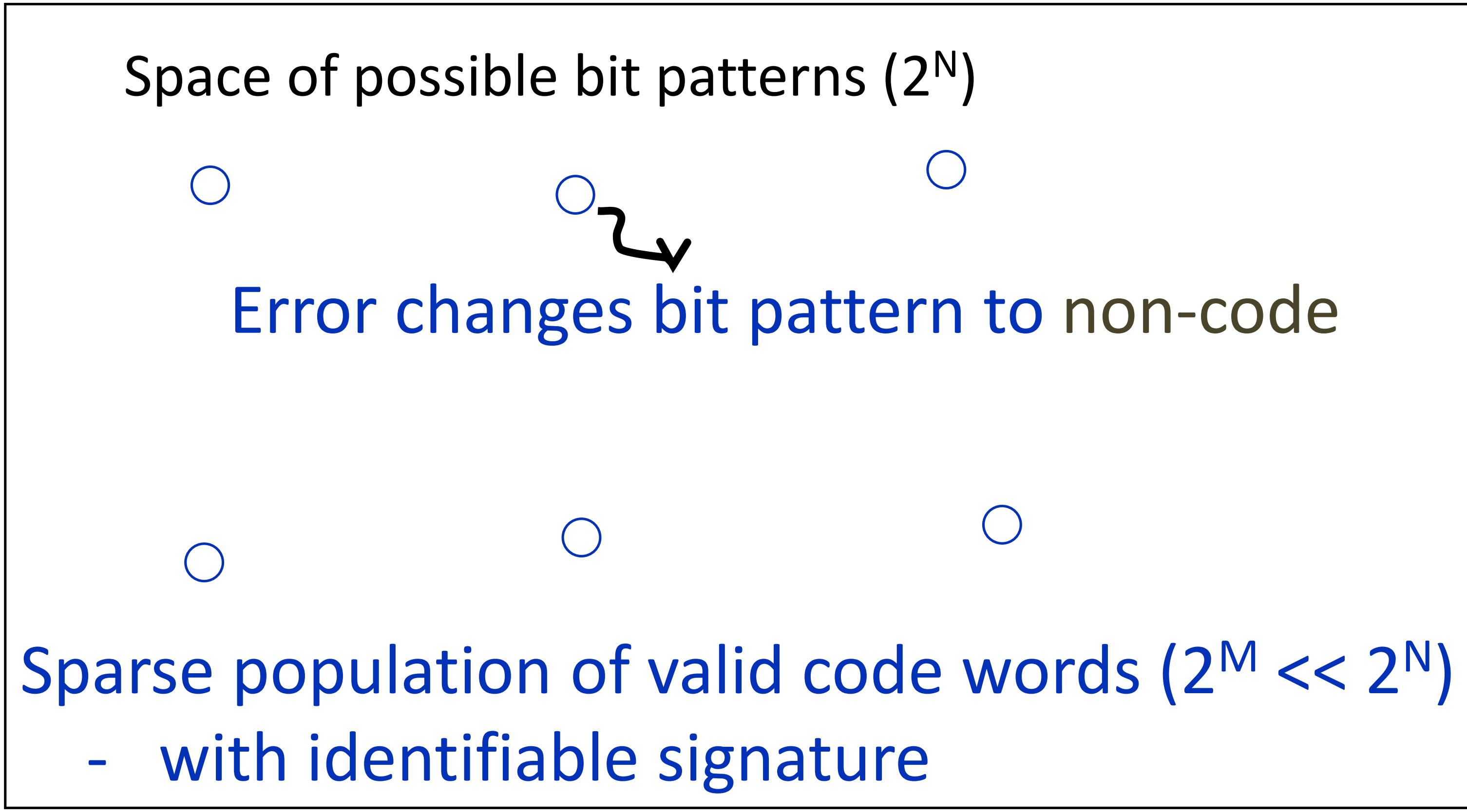
Parity Example

- 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?

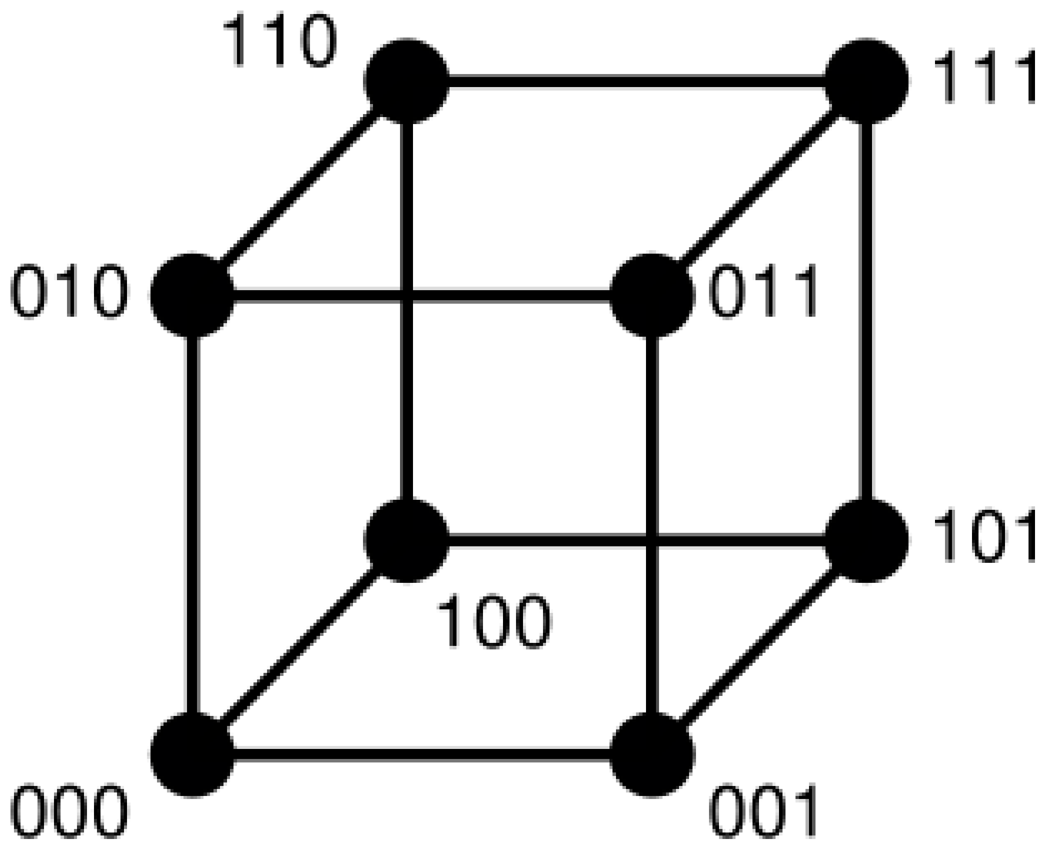
- Richard Hamming came up with simple to understand mapping to allow Error Correction at minimum distance of 3
  - Single error correction, double error detection
- Called "Hamming ECC"
  - Worked weekends on relay computer with unreliable card reader, frustrated with manual restarting
  - Got interested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

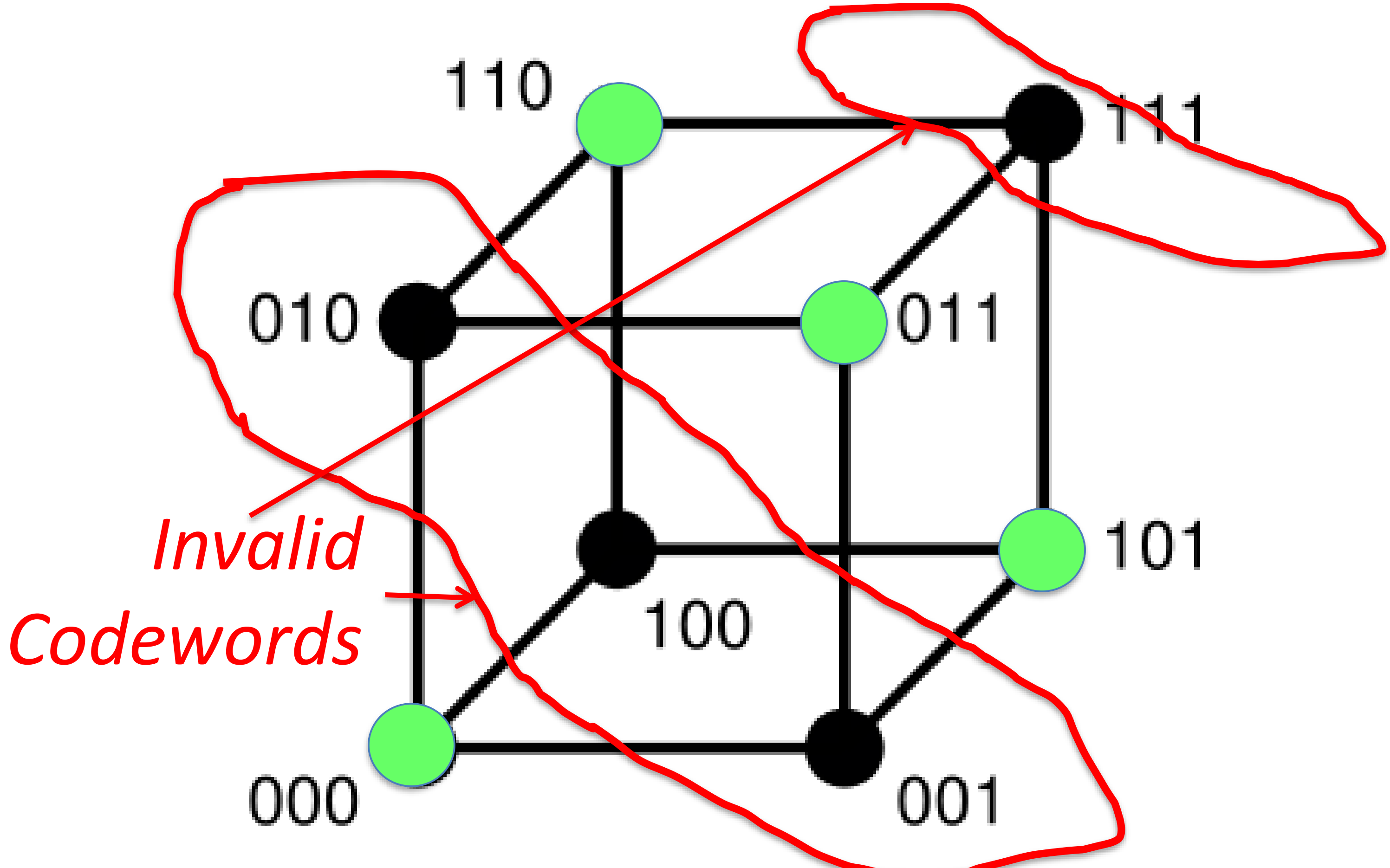


- Detection: bit pattern fails codeword check
- Correction: map to nearest valid code word

Hamming Distance: 8 code words



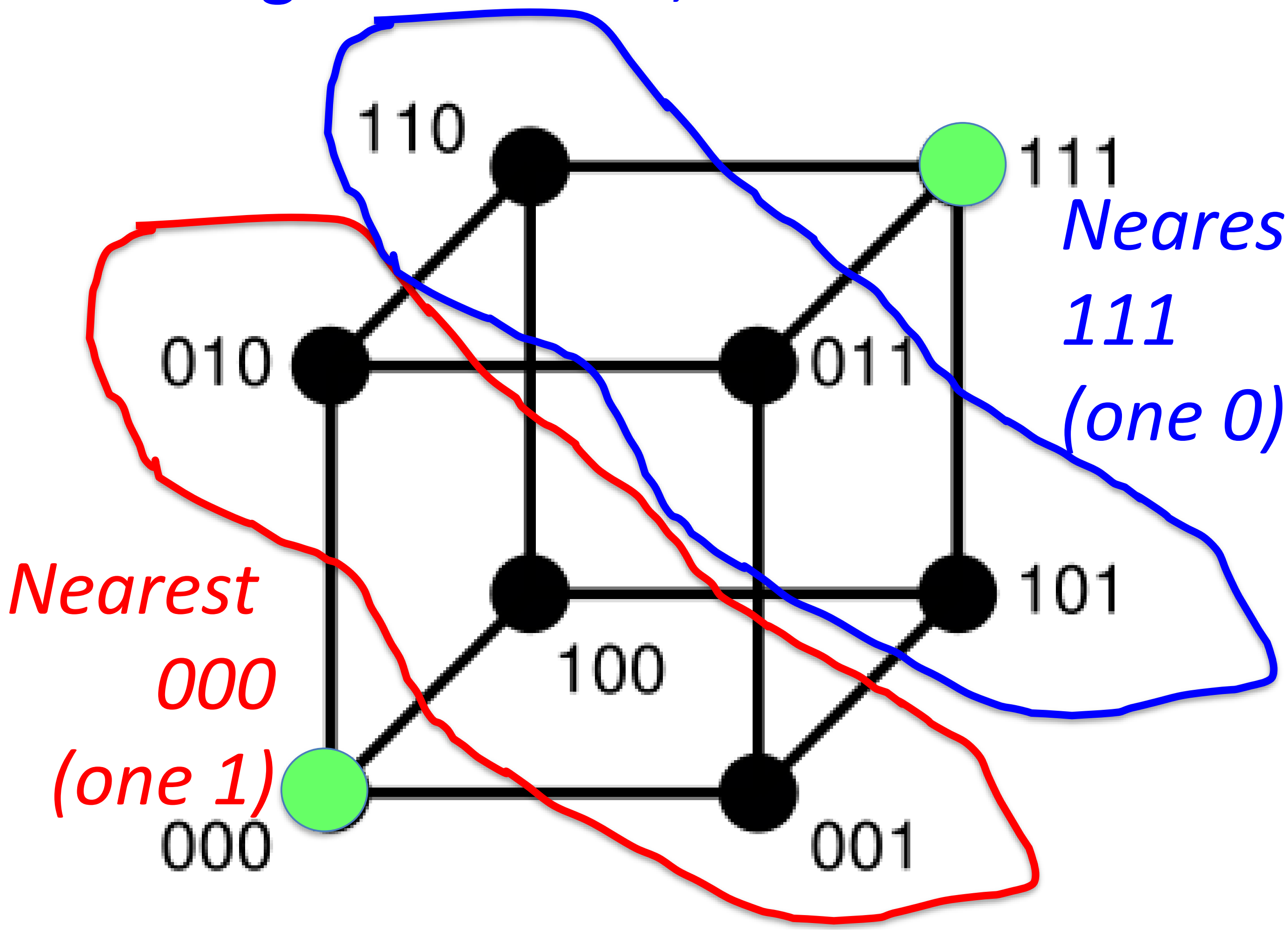
Hamming Distance 2: Detection
Detect Single Bit Errors



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

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

Hamming Error Correction Code

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

1. Mark all bit positions 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 data bits (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, ...)
3. Each data bit is covered by 2 or more parity bits

Hamming ECC

4. The position of parity bit determines sequence of data bits that it checks

- Bit 1 (0001_2): checks bits (1,3,5,7,9,11,...)
  - Bits with least significant bit of address = 1
- Bit 2 (0010_2): checks bits (2,3,6,7,10,11,14,15,…)
  - Bits with 2^{nd} least significant bit of address = 1
- Bit 4 (0100_2): checks bits (4-7, 12-15, 20-23, ...)
  - Bits with 3^{rd} least significant bit of address = 1
- Bit 8 (1000_2): checks bits (8-15, 24-31, 40-47 ,...)
  - Bits with 4^{th} least significant bit of address = 1

Graphic of 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

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

Hamming ECC

5. 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
- Calculate the parity bits

Hamming ECC

_ _ 1 _ 0 0 1 _ 1 0 1 0

- Position 1 checks bits 1, 3, 5, 7, 9, 11:
  ? _ 1 _ 0 0 **1 _ 1** 0 **1** 0. set position 1:
  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:
  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:
  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:
  - 0 1 1 1 0 0 1 **0 1 0 1 0**

Hamming ECC

- Final code word: $\underline{01}1\underline{1}001\underline{0}1010$
- Data word: 1 001 1010

Hamming ECC Error Check

- Suppose receive

  011100101110

  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

  $\underline{01}1\underline{1}001\underline{0}1110$

  $\underline{0}\ \ 1\ \ 0\ \ 1\ \ 1\ \ 1$ √

  $\underline{1}1\ \ \ 01\ \ \ 11$ X-Parity 2 in error

  $\underline{1}001\ \ \ \ \ 0$ √

  $\underline{0}1110$ X-Parity 8 in error

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

  $\underline{01}1\underline{1}001\underline{0}1\mathbf{1}10$

Hamming ECC Error Correct

- Flip the incorrect bit …

```
0 1 1 1 0 0 1 0 1 0 1 0
```

- Double check

```
0 1 1 1 0 0 1 0 1 0 1 0
0   1   0   1   1   1     √
  1 1     0 1     0 1     √
      1 0 0 1         0   √
              0 1 0 1 0   √
```

Hamming ECC Error Detect

- Suppose receive

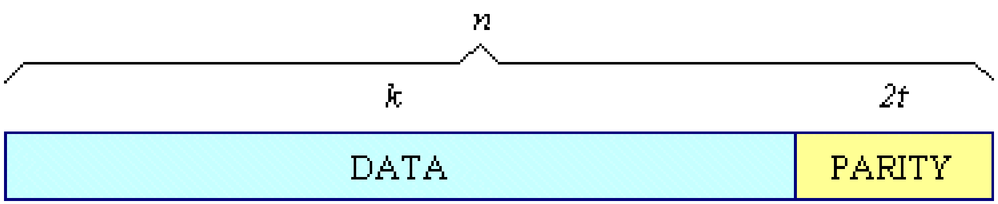
```
010100001010
0 0 0 0 1 1   √
 10  00  01   √
   1000    0  X
      01010   √
```

Two errors can be detected, but not correctable

How about ≥3 bits error?

Cyclic Redundancy Check

- Parity is not powerful enough to detect long runs of errors (also known as *burst errors*)
- Better Alternative: *Reed-Solomon Codes*
  - Used widely in CDs, DVDs, Magnetic Disks
  - RS(255,223) with 8-bit symbols: each codeword contains 255 code word bytes (223 bytes are data and 32 bytes are parity)



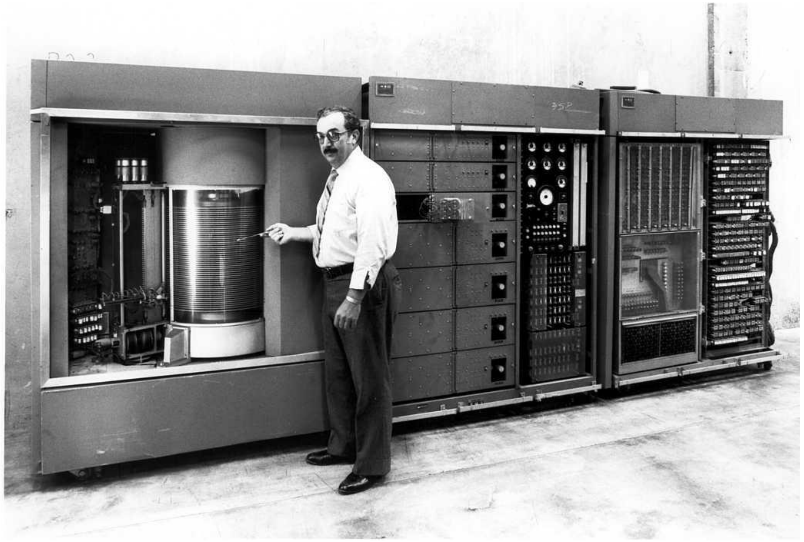
  - For this code: n = 255, k = 223, s = 8, 2t = 32, t = 16
  - Decoder can correct any errors in up to 16 bytes anywhere in the codeword

RAID: Redundancy for Disks

- Why we still worry about disks?
  - Trade-off: price, capacity, density, etc.
  - When you need storage space in petabytes (PB) or exabytes (EB)
    - 1 PB = 1024 TB
    - 1 EB = 1024 PB
  - Do not forget that flash-based SSDs also fail
    - Limited program/erase cycles $\Leftarrow$ wear leveling

Evolution of the Disk Drive

IBM 3390K, 1986



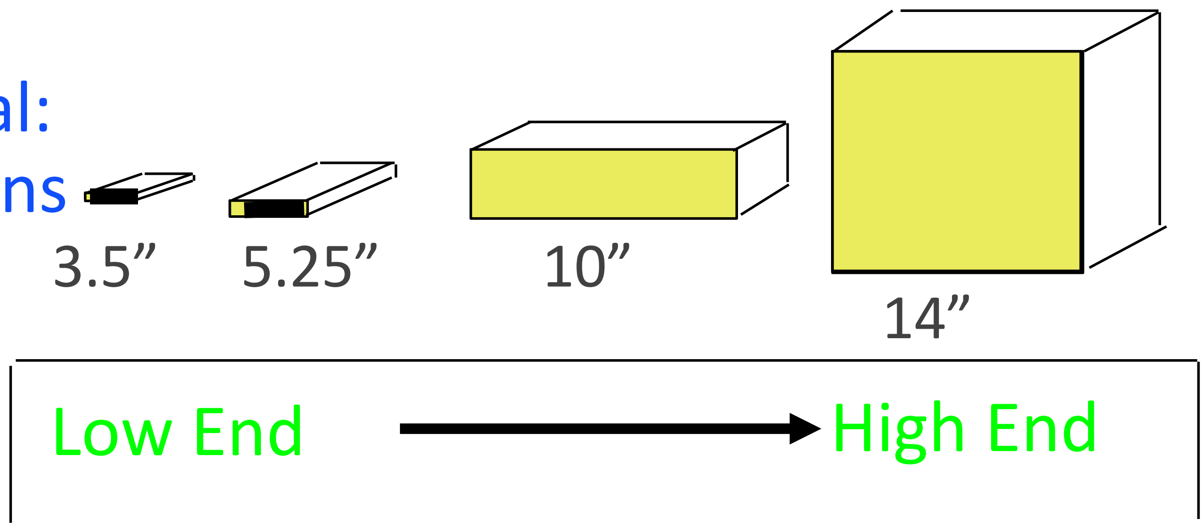
IBM RAMAC 305, 1956

Apple SCSI, 1986

Arrays of Small Disks

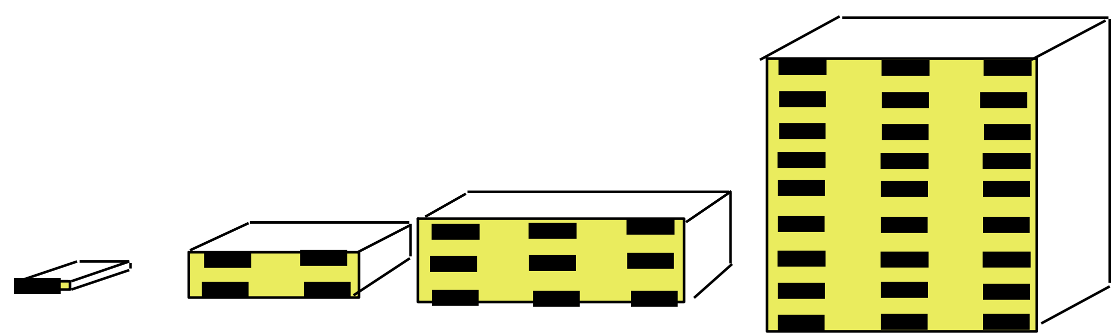
Can smaller disks be used to close gap in performance between disks and CPUs?

Conventional:
4 disk designs



Disk Array:
1 disk design

3.5”



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	

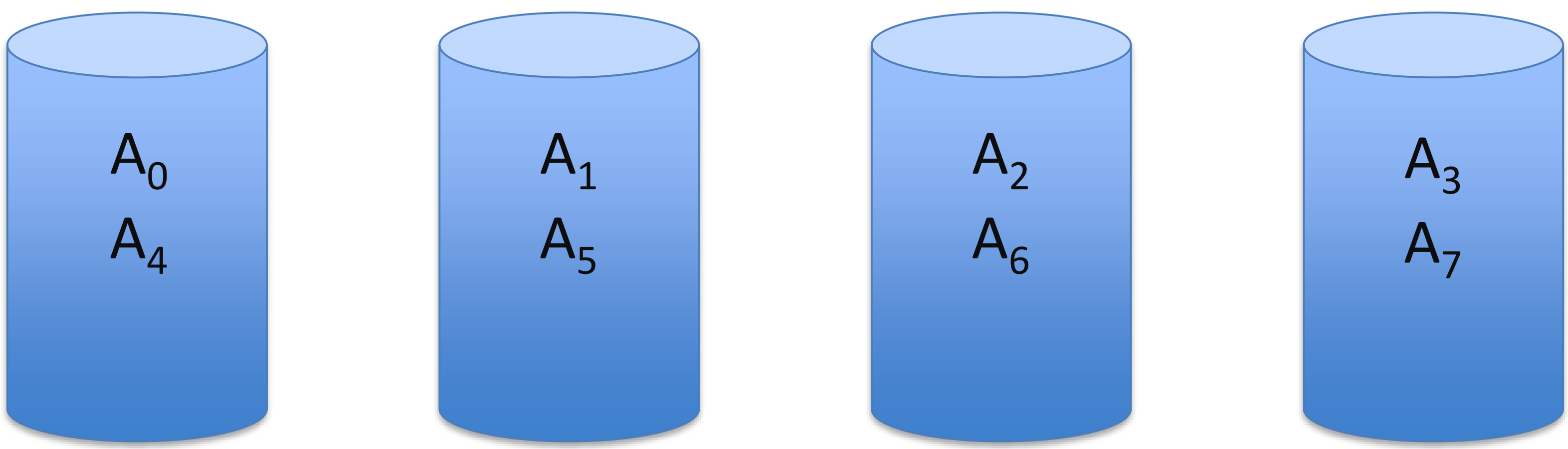
Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, <u>but what about reliability?</u>

RAID: Redundant Arrays of (Inexpensive) Disks

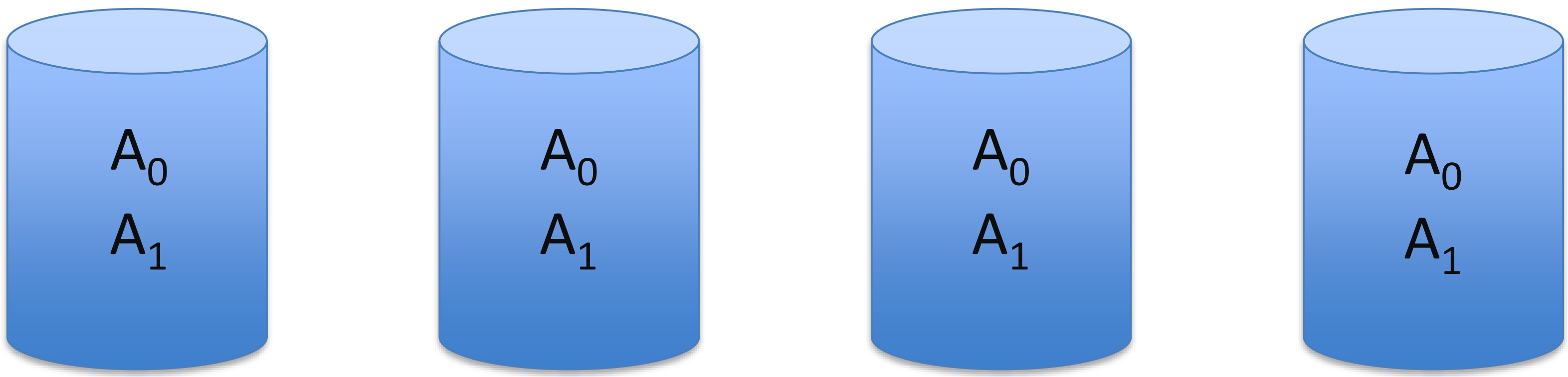
- Files are “striped” across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - ➔ Capacity penalty to store redundant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

RAID 0: Striping

- RAID 0 provides **no** fault tolerance or redundancy
  - Striping, or disk spanning
  - High performance



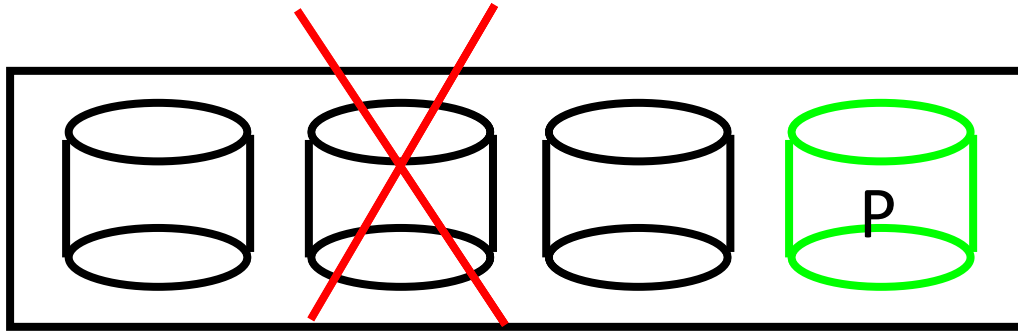
RAID 1: Disk Mirroring/Shadowing



- Each disk is fully duplicated onto its "mirror(s)"
  - Very high availability can be achieved
- Bandwidth sacrifice on write:
  - Logical write = N physical writes
  - Reads may be optimized
- Most expensive solution: 100% capacity overhead
- RAID 10 (striped mirrors), RAID 01 (mirrored stripes):
  - Combinations of RAID 0 and 1.

RAID 3: Parity Disk

10010011
11001101
10010011
. . .

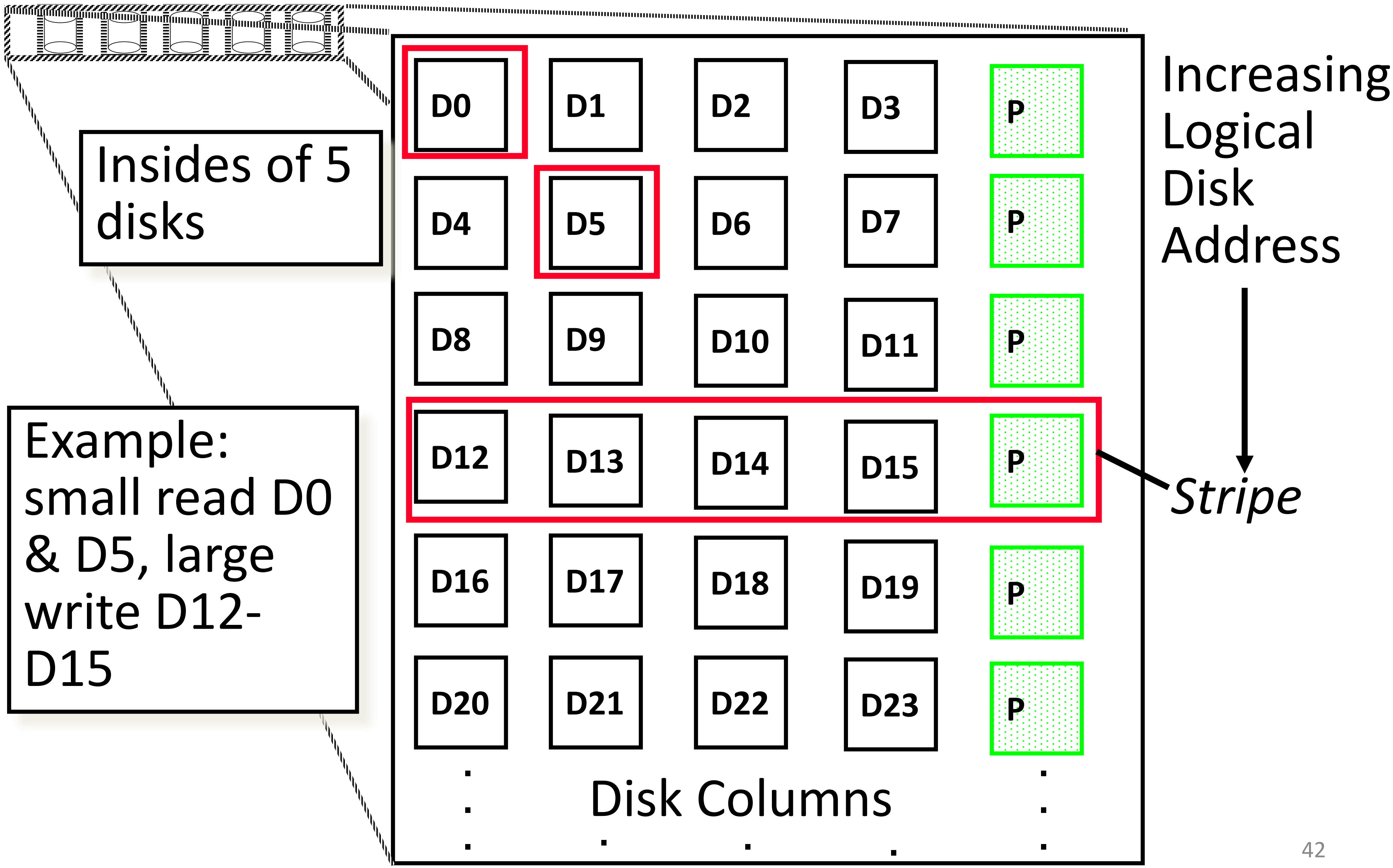


1	1	1	1
0	1	0	1
1	0	1	0
0	0	0	0
0	1	0	1
0	1	0	1
1	0	1	0
1	1	1	1

logical record
Striped physical records

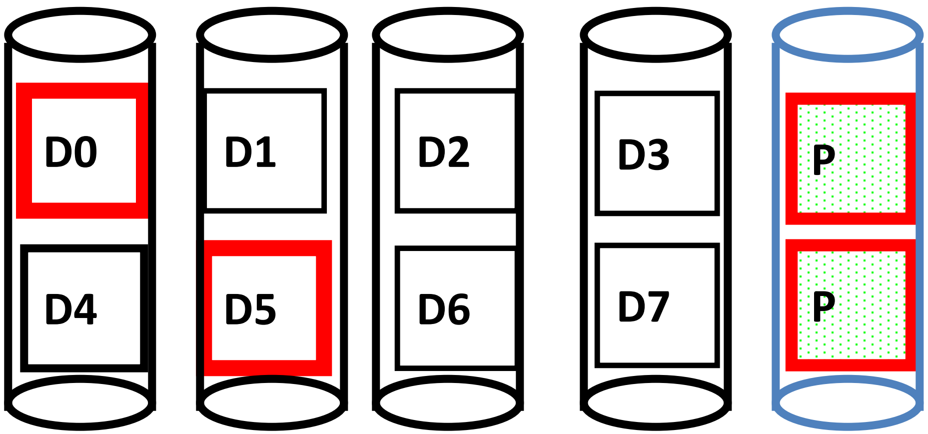
P contains sum of other disks per stripe mod 2 ("parity")
If disk fails, subtract P from sum of other disks to find missing information

RAID 4: High I/O Rate Parity



Inspiration for RAID 5

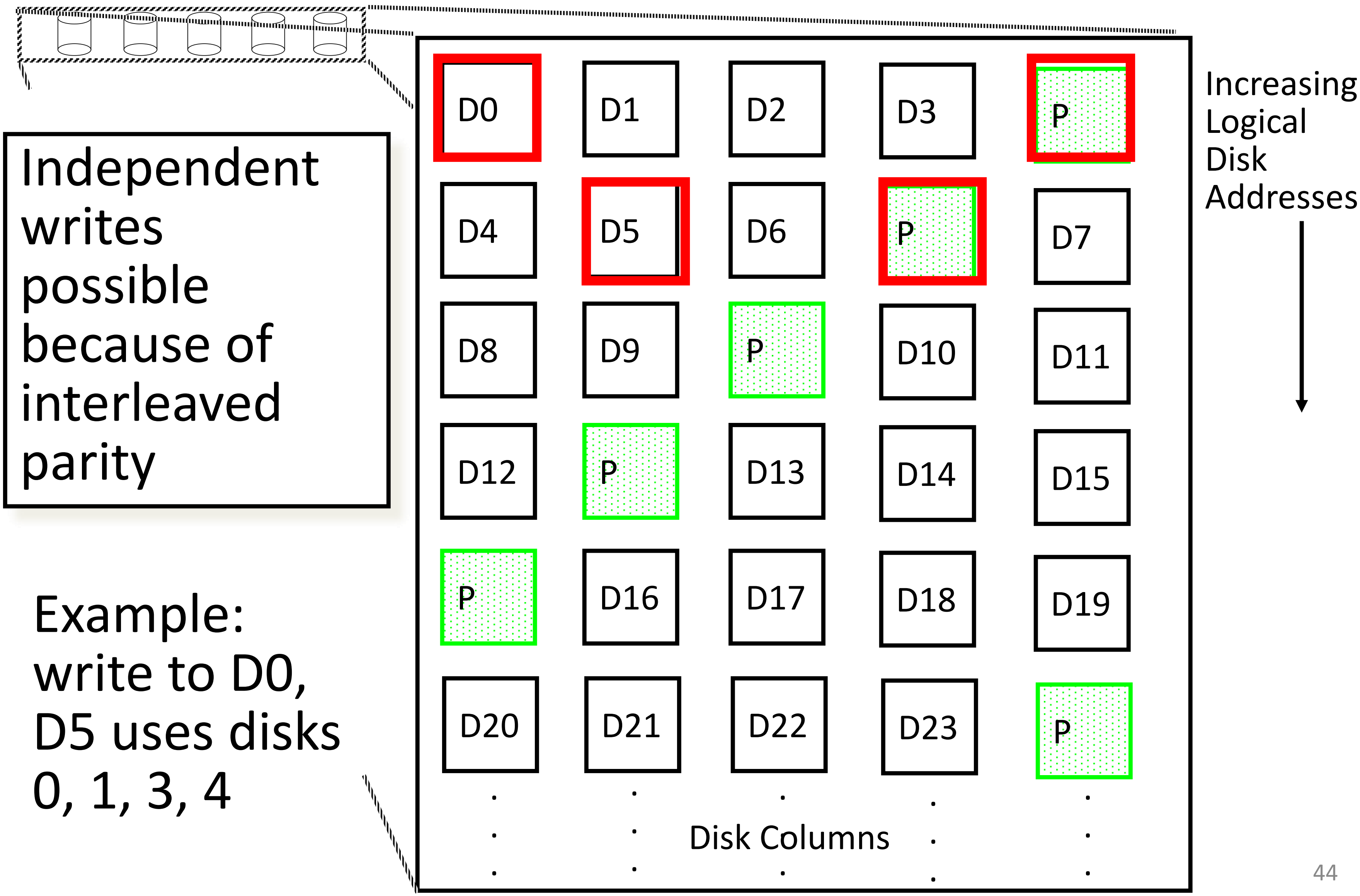
- RAID 4 works well for small reads
- Small writes (write to one disk):
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Small writes are limited by Parity Disk: Write to D0, D5 both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



RAID 5: High I/O Rate Interleaved Parity

Independent writes possible because of interleaved parity

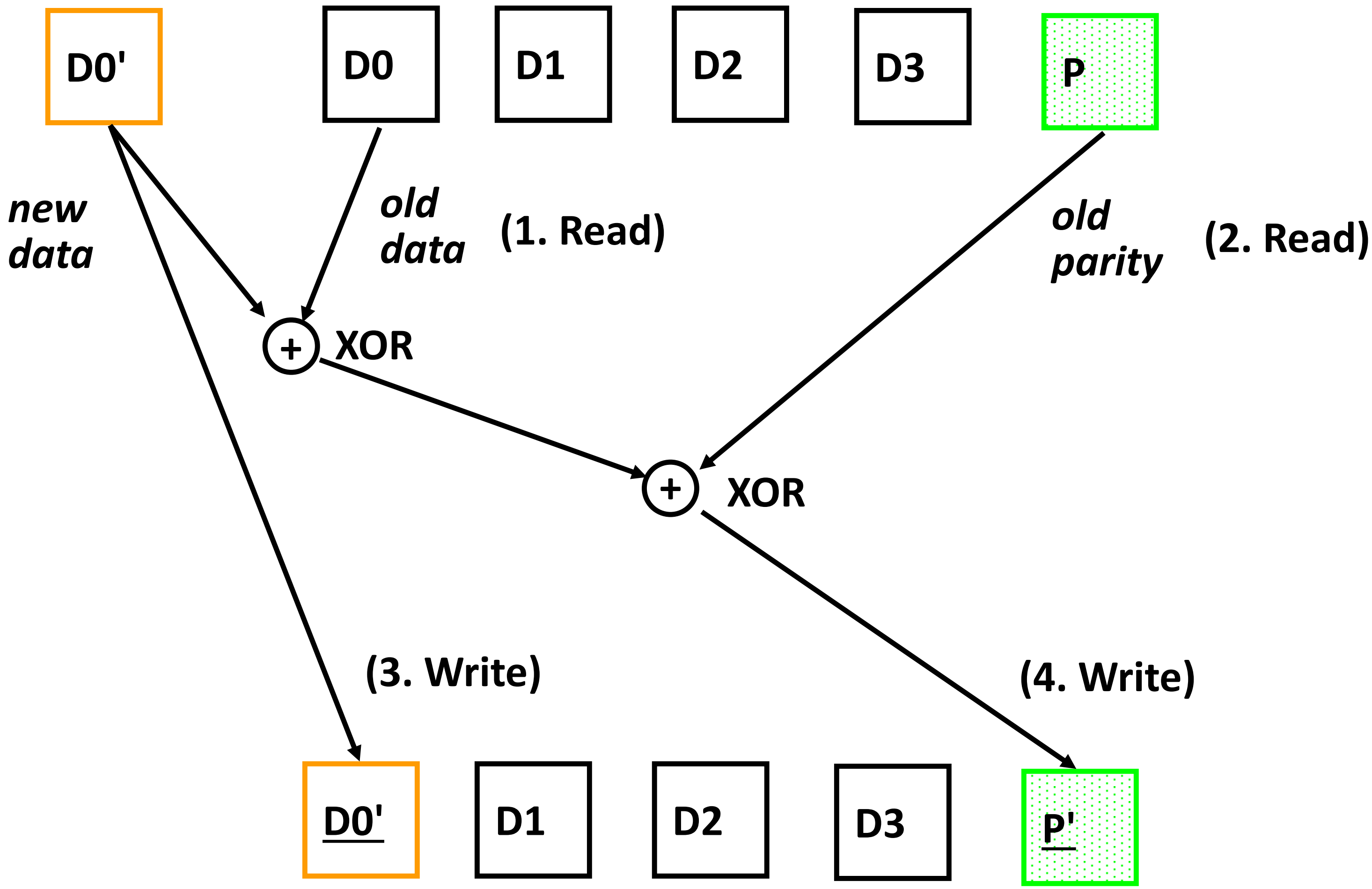
Example: write to D0, D5 uses disks 0, 1, 3, 4



Problems of Disk Arrays: Small Writes

RAID-5: Small Write Algorithm

1 Logical Write = 2 Physical Reads + 2 Physical Writes



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
- Memory
  - Hamming ECC: correct single, detect double
- RAID
  - Interleaved data and parity