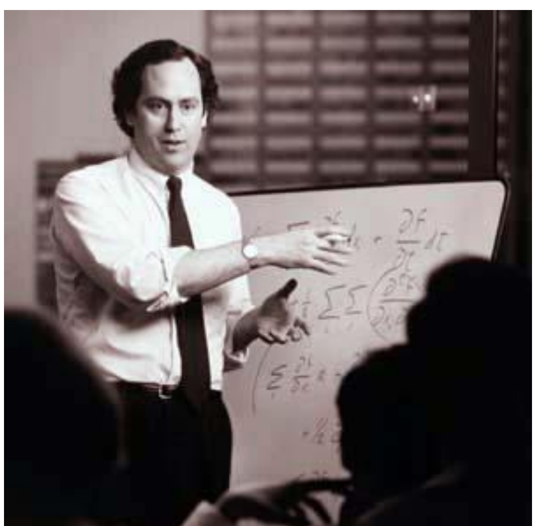
The magic of error-correcting codes

John MacCormick, Dickinson College

"Two weekends in a row I came in and found that all my stuff had been dumped and nothing was done. I was really aroused and annoyed because I wanted those answers and two weekends had been lost. And so I said, 'Dammit, if the machine can detect an error, why can't it locate the position of the error and correct it?' "



– Richard Hamming,
Bell Telephone Company, 1940s

The problem: computers need to store and transmit information using error-prone mechanisms, without making *any* mistakes

- Analogy: a phone number is useless if even one digit is wrong
- Realistic example:
  - 100 MB software download
  - A single incorrect bit could make it crash and/or destroy your data
  - Therefore, even 99.999999% accuracy is not good enough

What causes the errors?

- Examples:
  - WiFi has interfering and competing signals
  - Magnetic media on a hard drive can be unreliable
  - Copper wire and optical fiber can suffer from noise
  - CDs and DVDs can have scratches and dust
- In fact, every known method of storing or transmitting information is subject to errors

The problem: computers need to store and transmit information using error-prone mechanisms, without making *any* mistakes

- Solutions (the main topic of this talk):
  - Error-*detecting* codes
  - Error-*correcting* codes
  - *Erasure* codes

Plan of attack for understanding error-detecting/correcting/erasure codes

- Part A: 5 tricks
  - Each one is unrealistically naïve, but gives insight into how real-world codes work

- Part B: 3 interesting applications

Trick 1: the repetition trick

- Example: receive bank balance of $5293.75. Is it correct?
- Simple fix: ask them to send it four more times:

transmission 1:	$	5	2	9	3	.	7	5
transmission 2:	$	5	2	1	3	.	7	5
transmission 3:	$	5	2	1	3	.	1	1
transmission 4:	$	5	4	4	3	.	7	5
transmission 5:	$	7	2	1	8	.	7	5

- Choose the majority vote for each digit

Trick 1: the repetition trick

- Example: receive bank balance of \$5293.75. Is it correct?
- Simple fix: ask them to send it four more times:

transmission 1:	\$	5	2	9	3	.	7	5
transmission 2:	\$	5	2	1	3	.	7	5
transmission 3:	\$	5	2	1	3	.	1	1
transmission 4:	\$	5	4	4	3	.	7	5
transmission 5:	\$	7	2	1	8	.	7	5
most common digit:	\$	5	2	1	3	.	7	5

- Choose the majority vote for each digit

Trick 1: the repetition trick

- Disadvantage: Enormous overhead e.g. 400% overhead for 4 extra repetitions
- Nevertheless, this “stupid” trick is widely used (for storage, not communication)
  - e.g. the Google file system stores 3 copies of each chunk (Ghemawat et al 2003)

Trick 2: the redundancy trick

- Main idea: transmit the bank balance using a redundant description of each digit
  - e.g. use English words:

    five two one three point seven five

  - Even with 20% random errors, it's unambiguous:

    `fiqe kwo one thrxp point sivpn fivq`

The redundancy trick translates *symbols* into *code words*, and back again

Encoding

1	$\rightarrow$	one
2	$\rightarrow$	two
3	$\rightarrow$	three
4	$\rightarrow$	four
5	$\rightarrow$	five

Decoding

five	$\rightarrow$	5	(exact match)
fiqe	$\rightarrow$	5	(closest match)
twe	$\rightarrow$	2	(closest match)

A code using English words for digits.

The redundancy trick is used in real computer systems

Encoding

0000	$\rightarrow$	0000000
0001	$\rightarrow$	0001011
0010	$\rightarrow$	0010111
0011	$\rightarrow$	0011100
0100	$\rightarrow$	0100110

Decoding

0010111	$\rightarrow$	0010	(exact match)
0010110	$\rightarrow$	0010	(closest match)
1011100	$\rightarrow$	0011	(closest match)

Part of the (7,4) Hamming code, invented in 1947.
Hamming-based codes are still used today, in DRAM.

Trick 3: the checksum trick

- Basic idea: message is a string of digits, checksum is the sum of the digits, mod 10.

						checksum
original message	4	6	7	5	6	8
message with one error	1	6	7	5	6	5
message with two errors	1	5	7	5	6	4
message with two (different) errors	2	8	7	5	6	8

- Simple checksum detects any *single* error, but does not necessarily detect multiple errors
- Fancier checksums are ubiquitous in real life (e.g. ethernet, TCP). Also closely related to hashes (e.g. MD5, SHA-256).

Trick 4: the staircase checksum

- Basic idea: to detect two errors, include a second checksum
- Compute 2nd checksum from a "staircase," e.g.
  - (1 x 1st digit) + (2 x 2nd digit) + (3 x 3rd digit) + …

						simple and staircase checksums
original message	4	6	7	5	6	8 7
message with one error	1	6	7	5	6	5 4
message with two errors	1	5	7	5	6	4 2
message with two (different) errors	2	8	7	5	6	8 9
message with two (again different) errors	6	5	7	5	6	9 7

- Oops, doesn't actually work, unless the staircase operations are in a certain finite field.
  - When done right, with multiple staircases, this gives Reed-Solomon codes, which are used in real life (e.g. on CDs)

Trick 5: the pinpoint trick

- Main idea: use horizontal and vertical checksums to pinpoint the error

4 8 3 7 5 4 3 6 2 2 5 6 3 9 9 7 ← raw message

4	8	3	7
5	4	3	6
2	2	5	6
3	9	9	7

Trick 5: the pinpoint trick

- Main idea: use horizontal and vertical checksums to pinpoint the error

raw message

4 8 3 7 5 4 3 6 2 2 5 6 3 9 9 7

4	8	3	7
5	4	3	6
2	2	5	6
3	9	9	7

transmitted message

checksum

4	8	3	7	2
5	4	3	6	8
2	2	5	6	5
3	9	9	7	8
4	3	0	6	

checksum

How to pinpoint the error

received message

4 8 3 7 2 5 4 3 6 8 2 7 5 6 5 3 9 9 7 8 4 3 0 6

4	8	3	7	2
5	4	3	6	8
2	7	5	6	5
3	9	9	7	8
4	3	0	6	

How to pinpoint the error

4 8 3 7 2 5 4 3 6 8 2 7 5 6 5 3 9 9 7 8 4 3 0 6 ← received message

4	8	3	7	2
5	4	3	6	8
2	7	5	6	5
3	9	9	7	8
4	3	0	6	

					received	computed
	4	8	3	7	2	2
	5	4	3	6	8	8
	2	7	5	6	5	0
	3	9	9	7	8	8
received	4	3	0	6		
computed	4	8	0	6		

How to pinpoint the error

4 8 3 7 2 5 4 3 6 8 2 7 5 6 5 3 9 9 7 8 4 3 0 6

received message

4	8	3	7	2
5	4	3	6	8
2	7	5	6	5
3	9	9	7	8
4	3	0	6	

received

computed

4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	0
3	9	9	7	8	8
4	3	0	6		
4	8	0	6		

received

computed

How to pinpoint the error

4 8 3 7 2 5 4 3 6 8 2 7 5 6 5 3 9 9 7 8 4 3 0 6

received message

4	8	3	7	2
5	4	3	6	8
2	7	5	6	5
3	9	9	7	8
4	3	0	6	

received computed

4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	0
3	9	9	7	8	8
4	3	0	6		
4	8	0	6		

received

computed

Summary of tricks

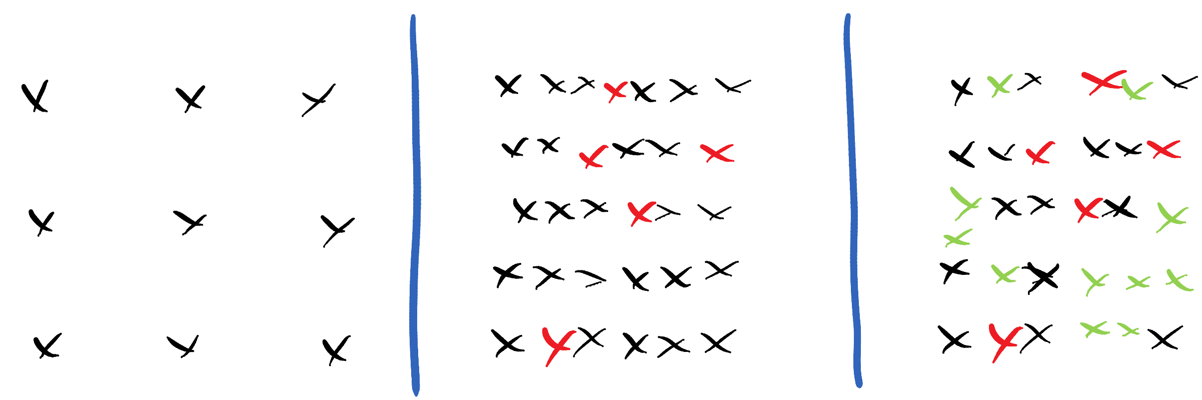
- Repetition: detects and corrects, but too much overhead.
- Redundancy: detects and corrects, but how do we find good codewords?
- Checksum: detects only. Can fill a single erasure.
- Multiple staircase checksums: detects and corrects multiple errors, and also good for erasures.
- Pinpoint: detects and corrects, but turns out to be less effective than state-of-the-art approaches.

Plan of attack for understanding error-detecting/correcting/erasure codes

- Part A: 5 tricks
  - Each one is unrealistically naïve, but gives insight into how real-world codes work
- Part B: 3 interesting applications

Application 1: how densely should we pack the bits on a disk?

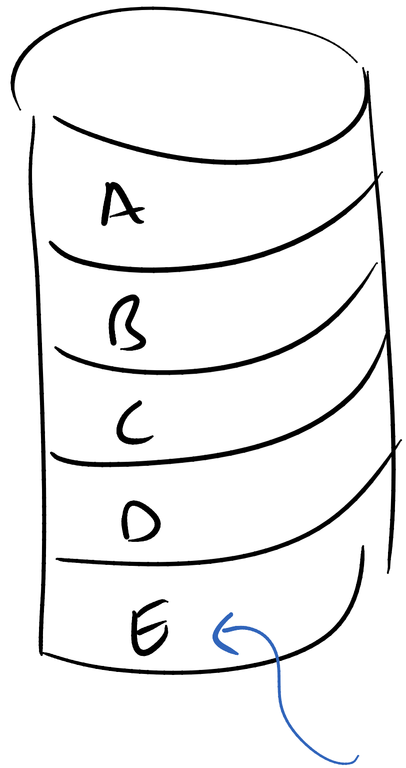
	Option A: Loose	Option B: Dense	Option C: Dense and redundant
raw density:	1	3	3
raw error rate:	10^{-10}	10^{-5}	10^{-5}
overhead:	0%	0%	50%
effective density:	1	3	2
effective error rate:	10^{-10}	10^{-5}	10^{-15}



Warning: numbers here are purely illustrative. Units omitted deliberately!

Application 2: disk arrays (RAID5 and RAID6)

RAID5

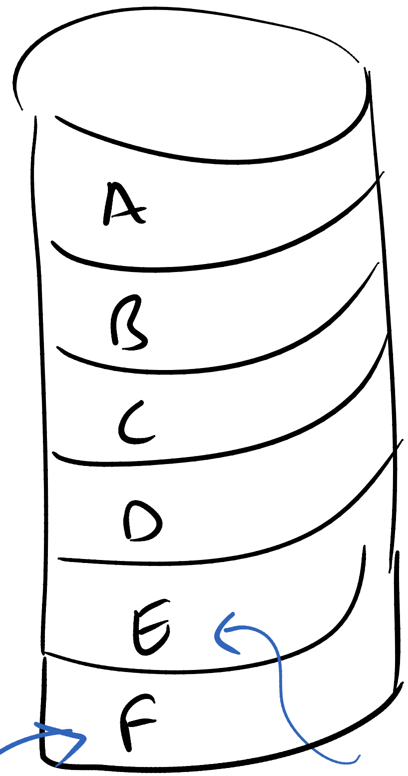


Can survive and rebuild after losing any *one* disk

$$E = A \oplus B \oplus C \oplus D$$

Application 2: disk arrays (RAID5 and RAID6)

RAID6



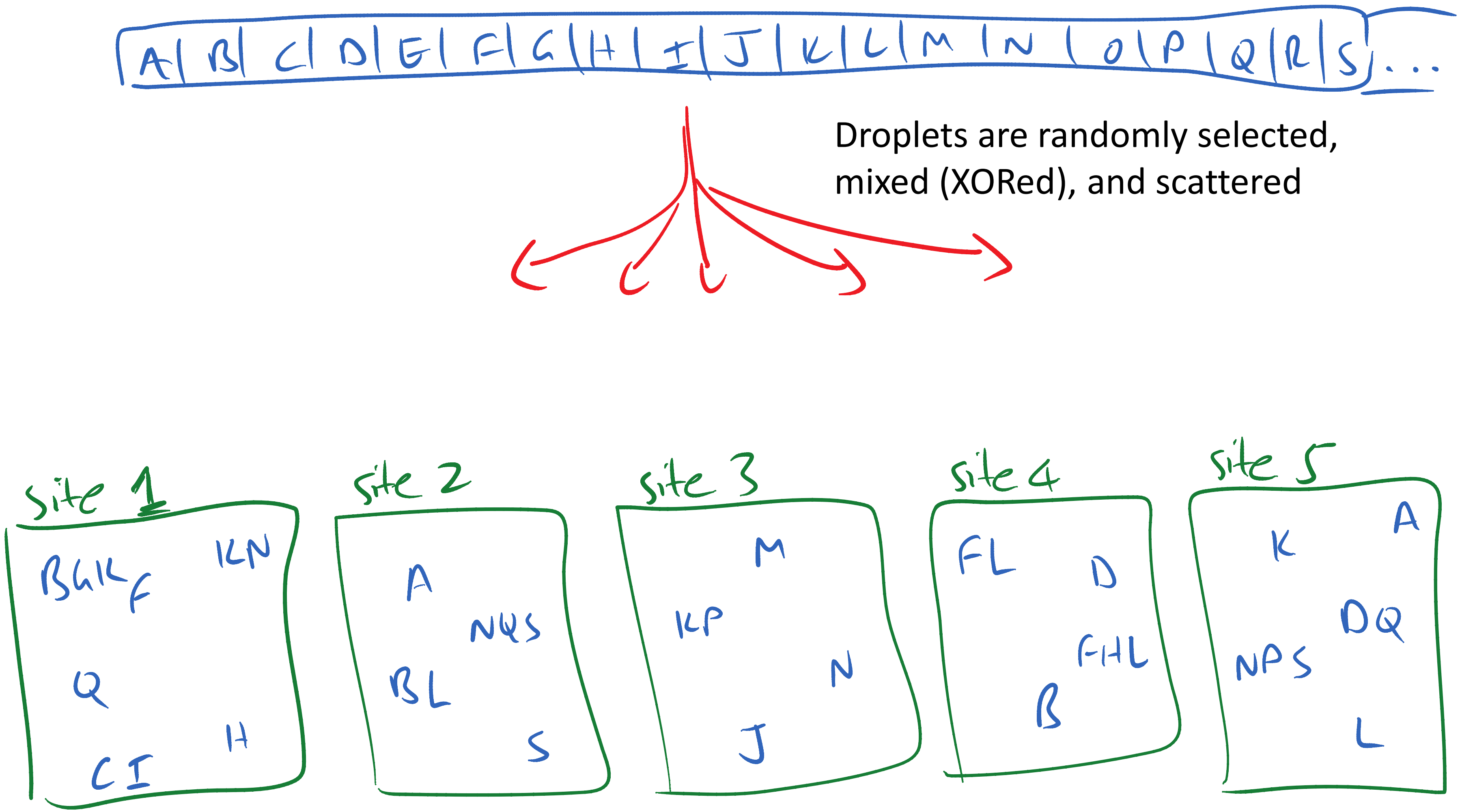
Can survive and rebuild after losing any *two* disks

$$E = A \oplus B \oplus C \oplus D$$

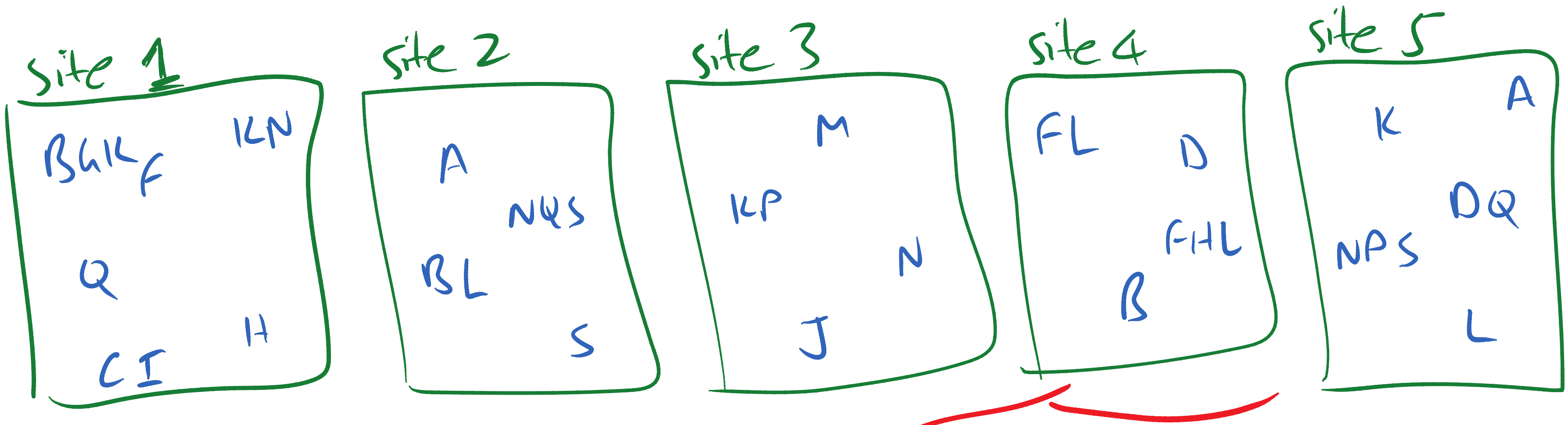
$$F = A \oplus 2B \oplus 3C \oplus 4D$$

Finite field operations, not ordinary arithmetic

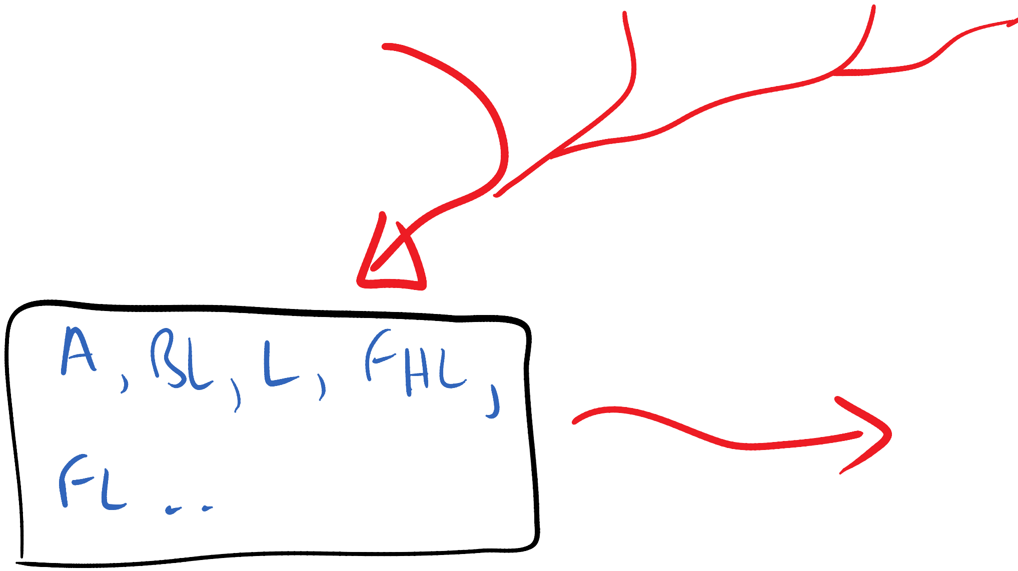
Application 3: fountain codes for geographically-distributed file storage

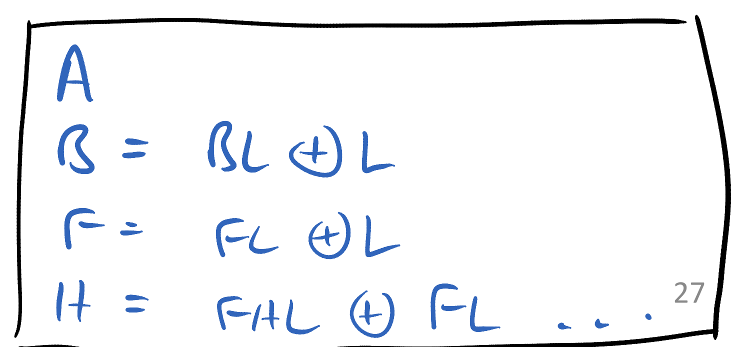


Application 3: fountain codes for geographically-distributed file storage

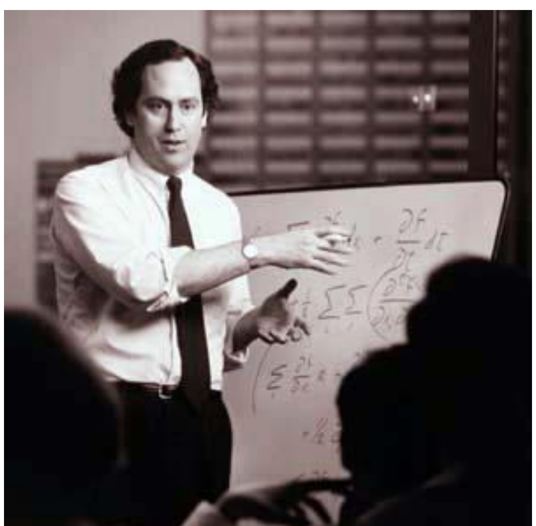


Mixed droplets are randomly gathered. Pure droplets can be reconstructed via XOR with probability ≈ 1, with about 5% overhead (Byers et al, 2002).





"Two weekends in a row I came in and found that all my stuff had been dumped and nothing was done. I was really aroused and annoyed because I wanted those answers and two weekends had been lost. And so I said, 'Dammit, if the machine can detect an error, why can't it locate the position of the error and correct it?' "



– Richard Hamming,
Bell Telephone Company, 1940s