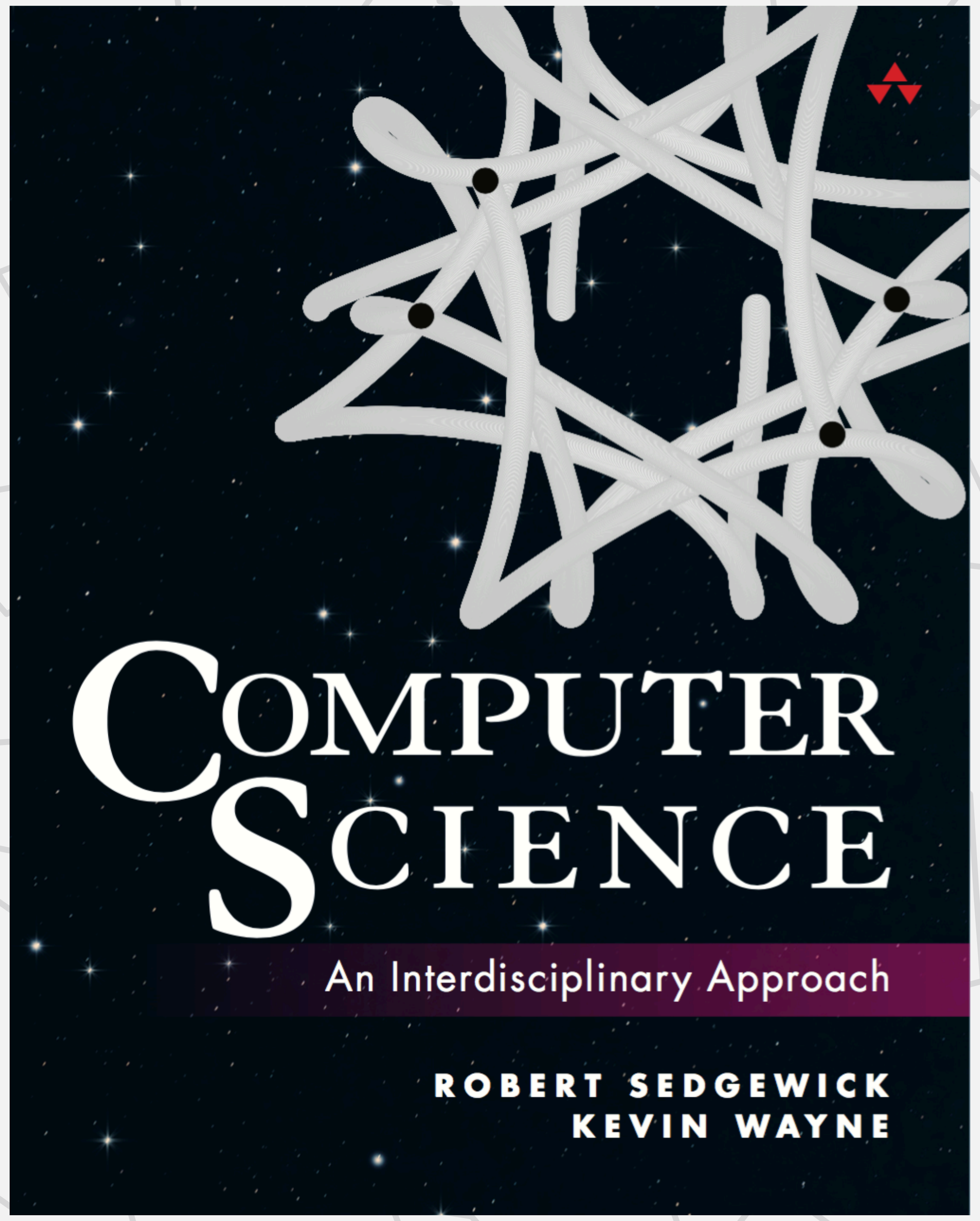


http://introcs.cs.princeton.edu

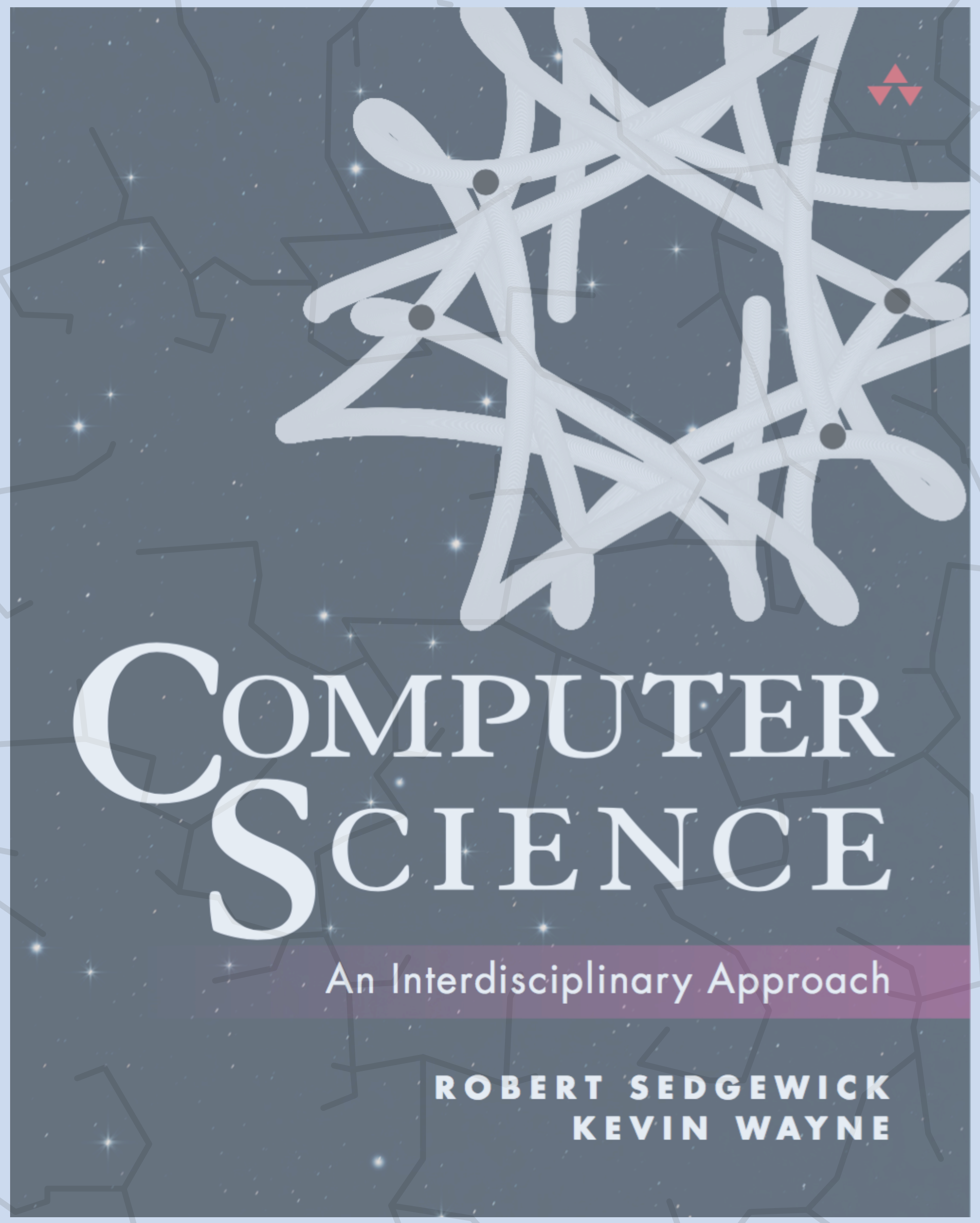
Hamming Codes in TOY

- *Hamming codes*
- *TOY simulator*
- *bugs to avoid*

Kevin Wayne
Last updated on 3/20/17 12:26 PM

Goals

- TOY: write two small machine-language programs.
- Hamming codes: learn about a widely used error-correcting code.

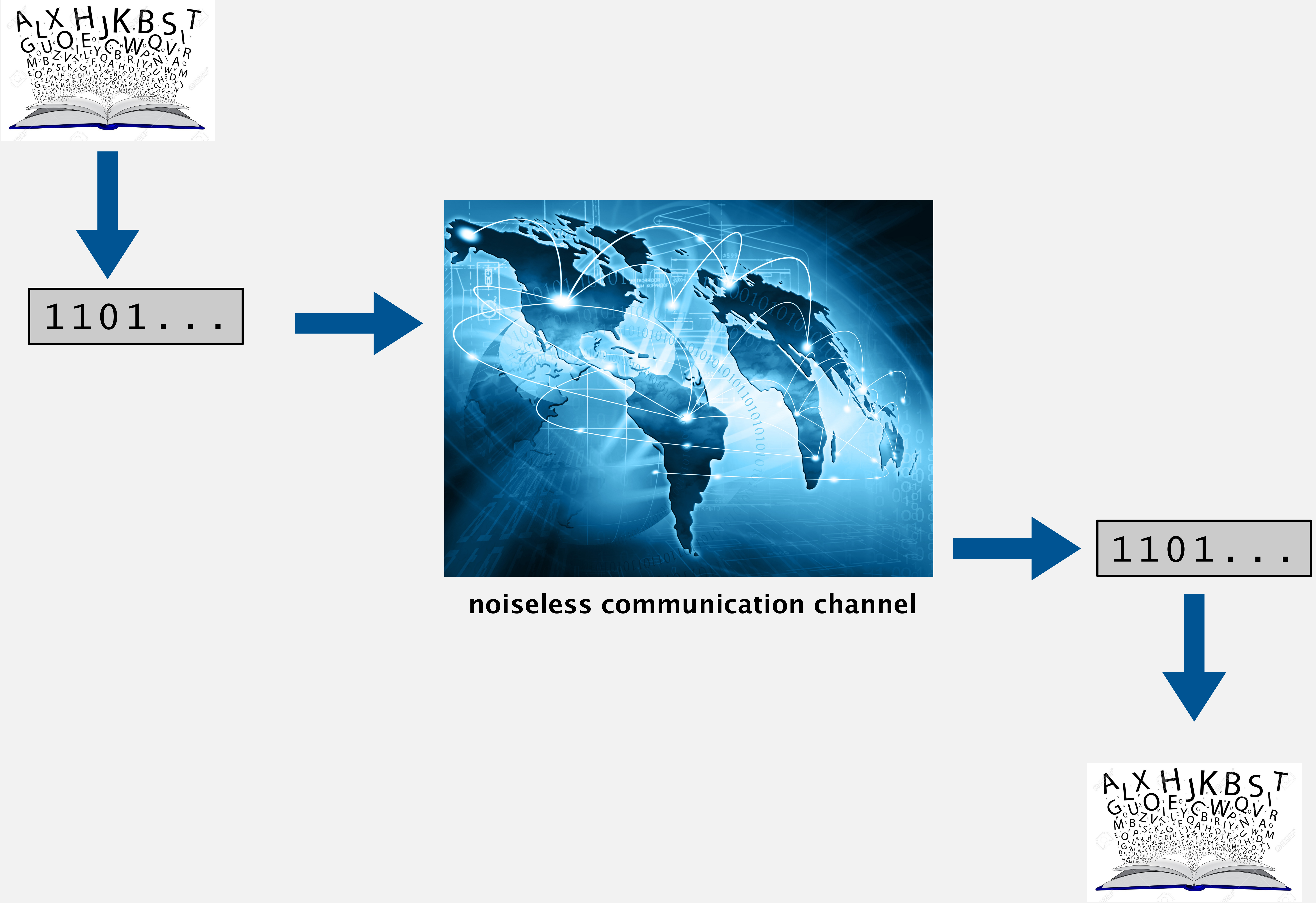


http://introcs.cs.princeton.edu

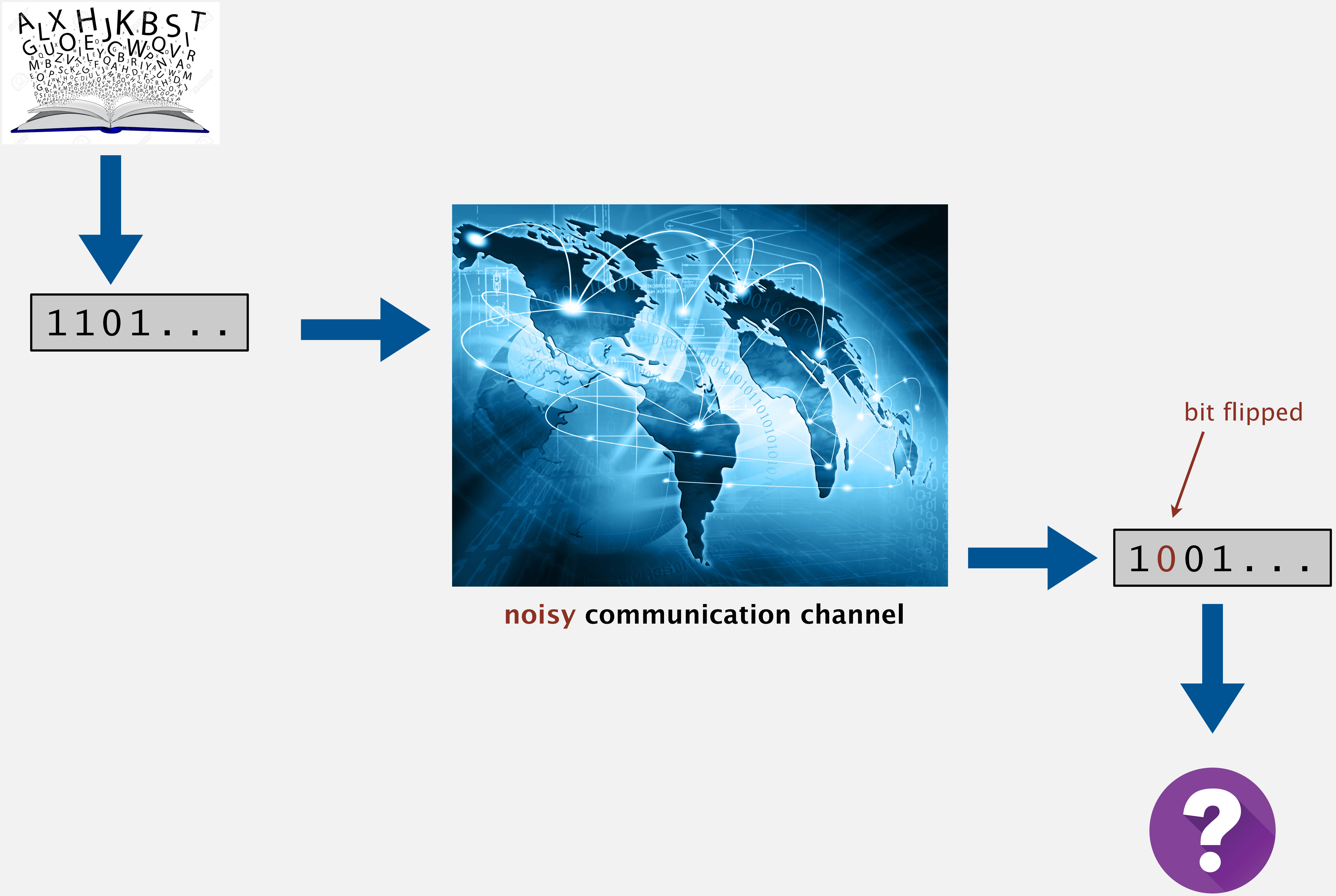
Hamming Codes in TOY

- *Hamming codes*
- *TOY simulator*
- *bugs to avoid*

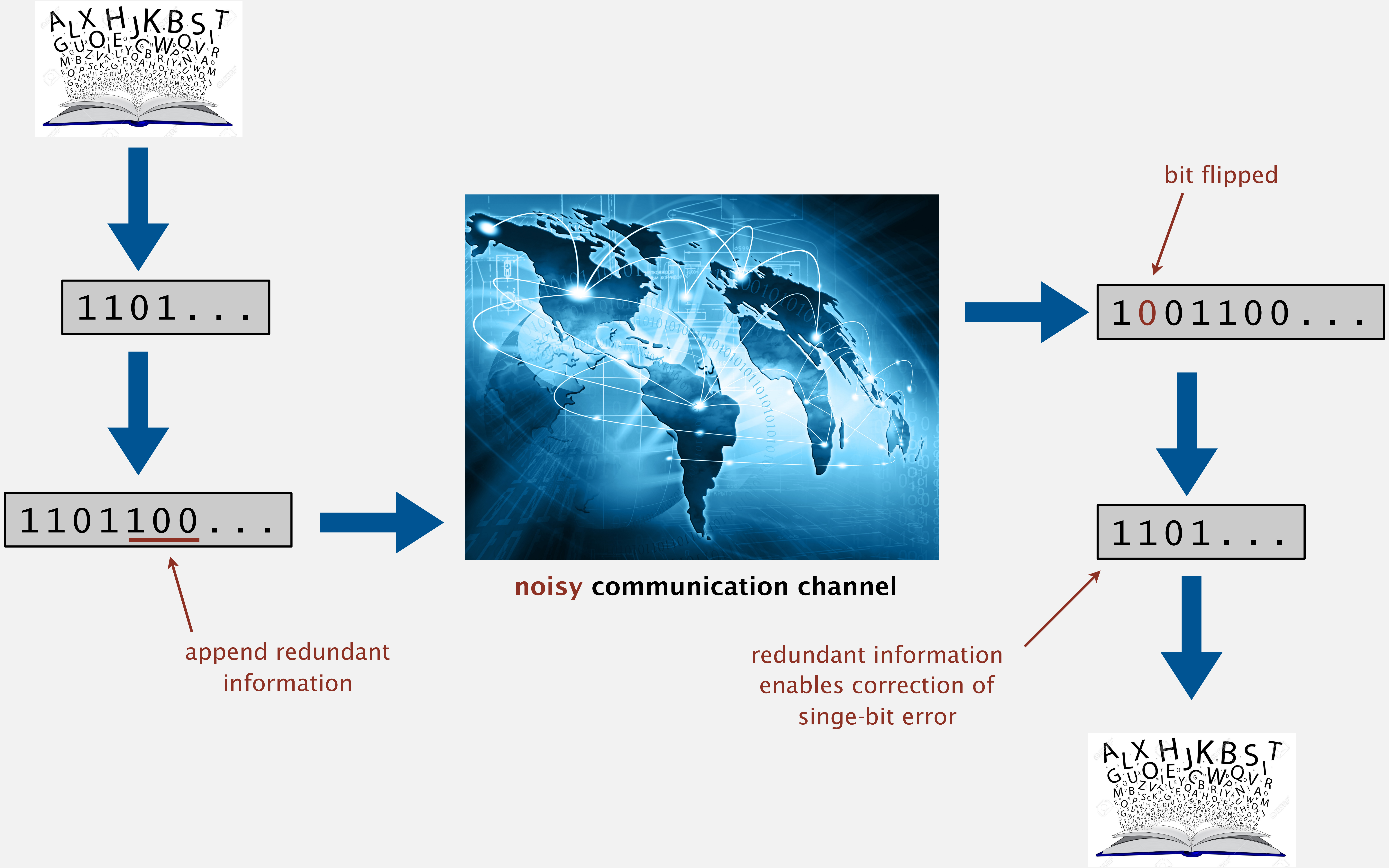
Noiseless communication channel



Noisy communication channel



Error-correcting codes



Error-correcting codes

Message bits: m_1, m_2, m_3, m_4.

Goal. Send and receive 4 message bits at a time.

Noiseless channel. What you send is what you receive.

Easy. Send m_1, m_2, m_3, m_4.

Noisy channel. One of the 4 bits might get flipped during transmission.

Attempt 1. Send m_1, m_2, m_3, m_4.

Attempt 2. Send $m_1, m_1, m_2, m_2, m_3, m_3, m_4, m_4$.

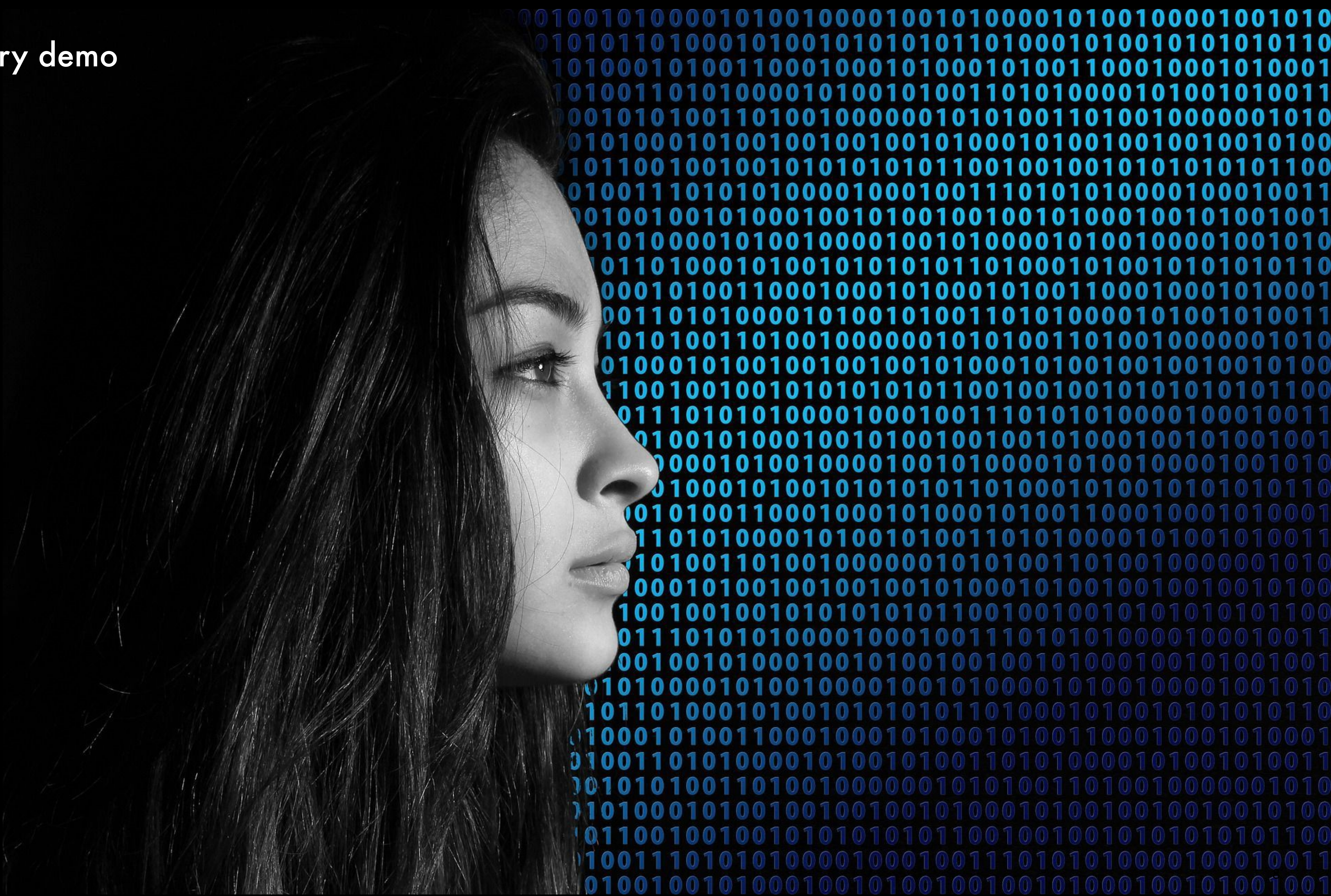
if two copies of m_4 are different, can detect error but not enough information to correct error

Attempt 3. Send $m_1, m_1, m_1, m_2, m_2, m_2, m_3, m_3, m_3, m_4, m_4, m_4$.

interpret m_4 as 1 if a majority of bits are 1; interpret m_4 as 0 if a majority of bits are 0

This assignment. 7–4 Hamming code: correct 1-bit errors, but using only 7 bits instead of 12.

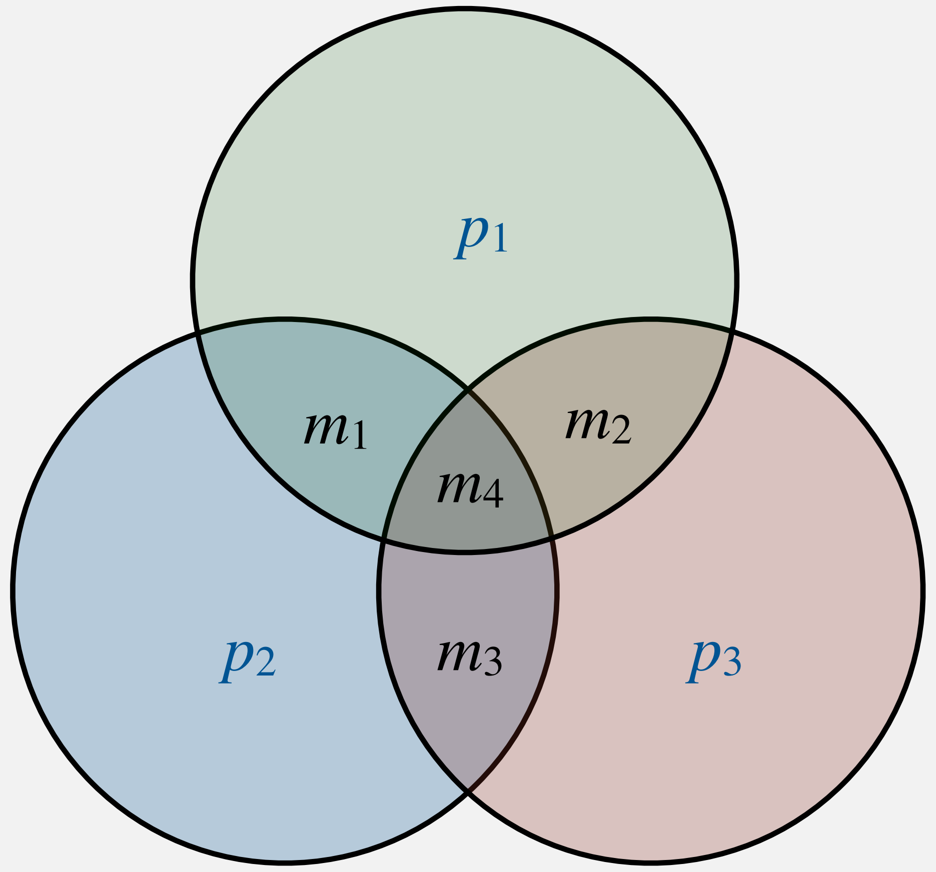
Madame Binary demo

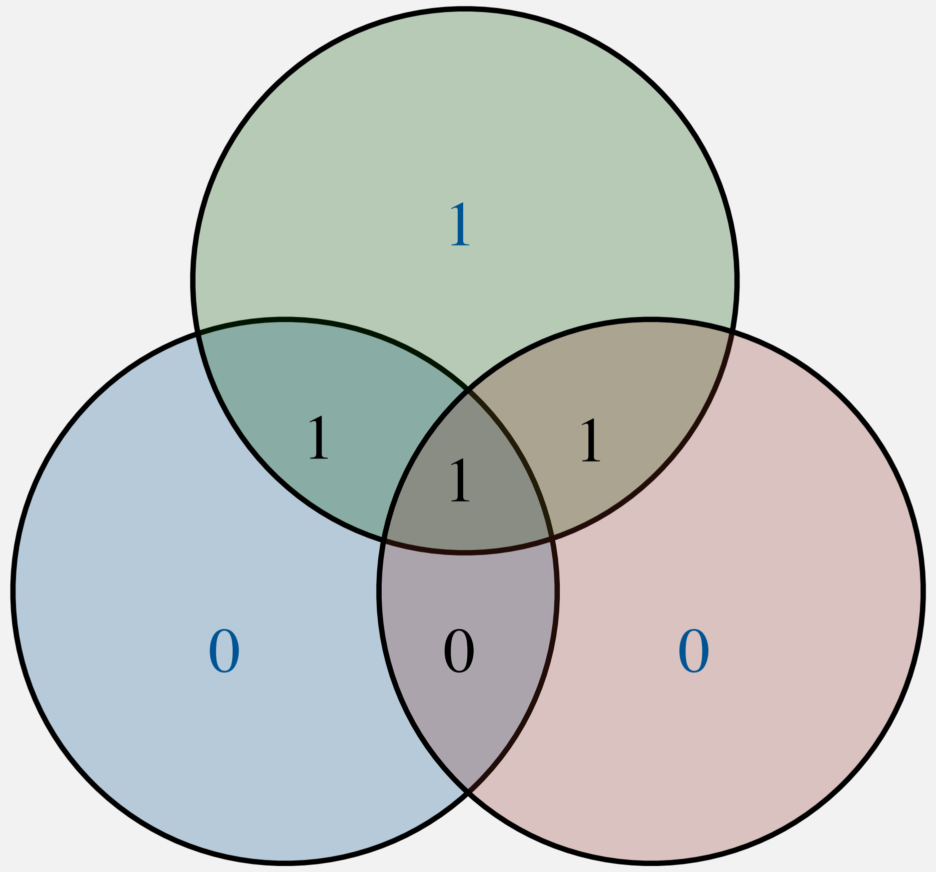


Parity bits

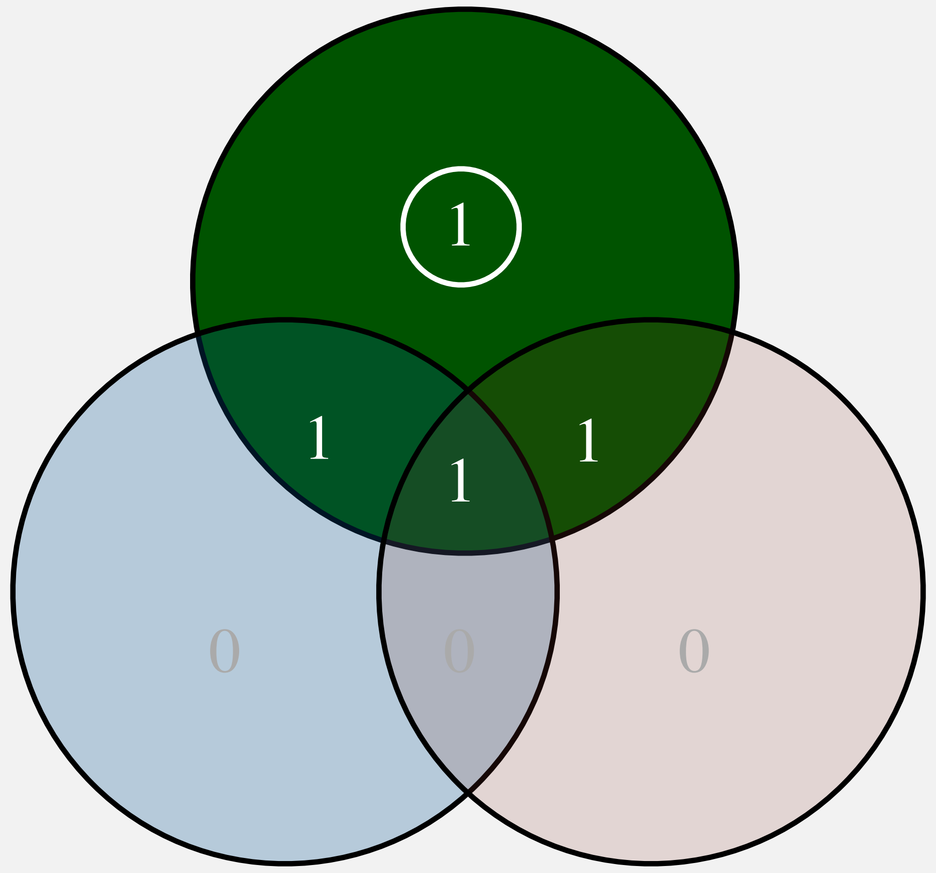
Message bits: m_1, m_2, m_3, m_4.

Parity bits: p_1, p_2, p_3.

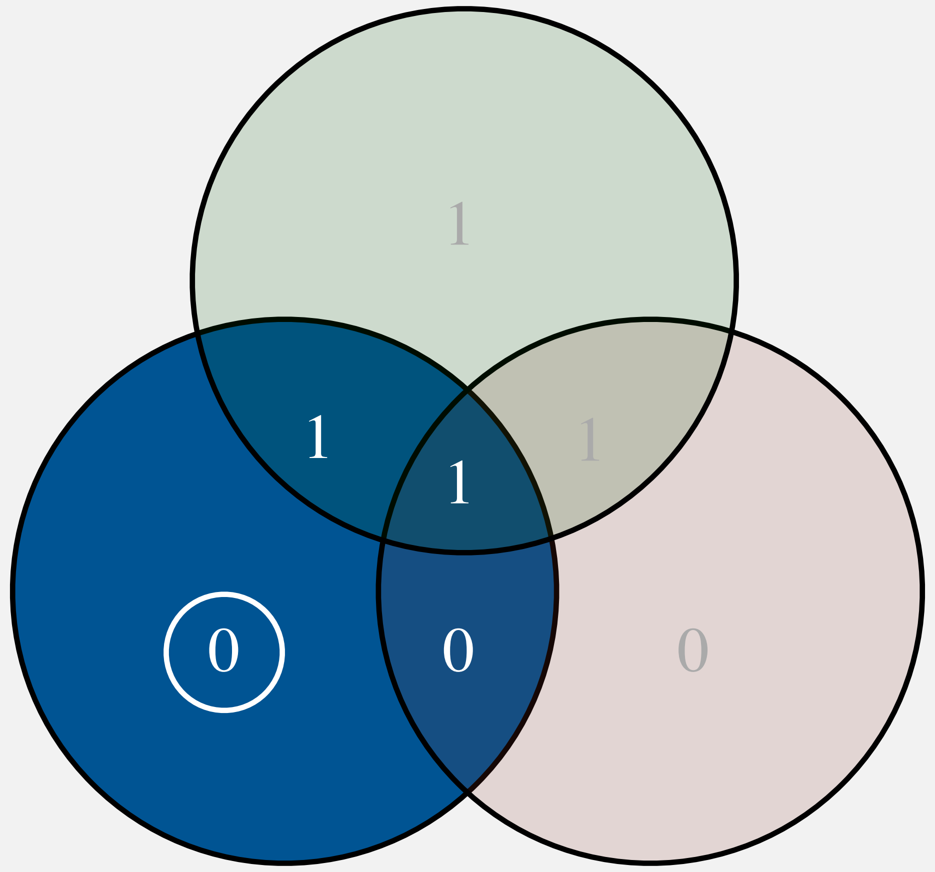




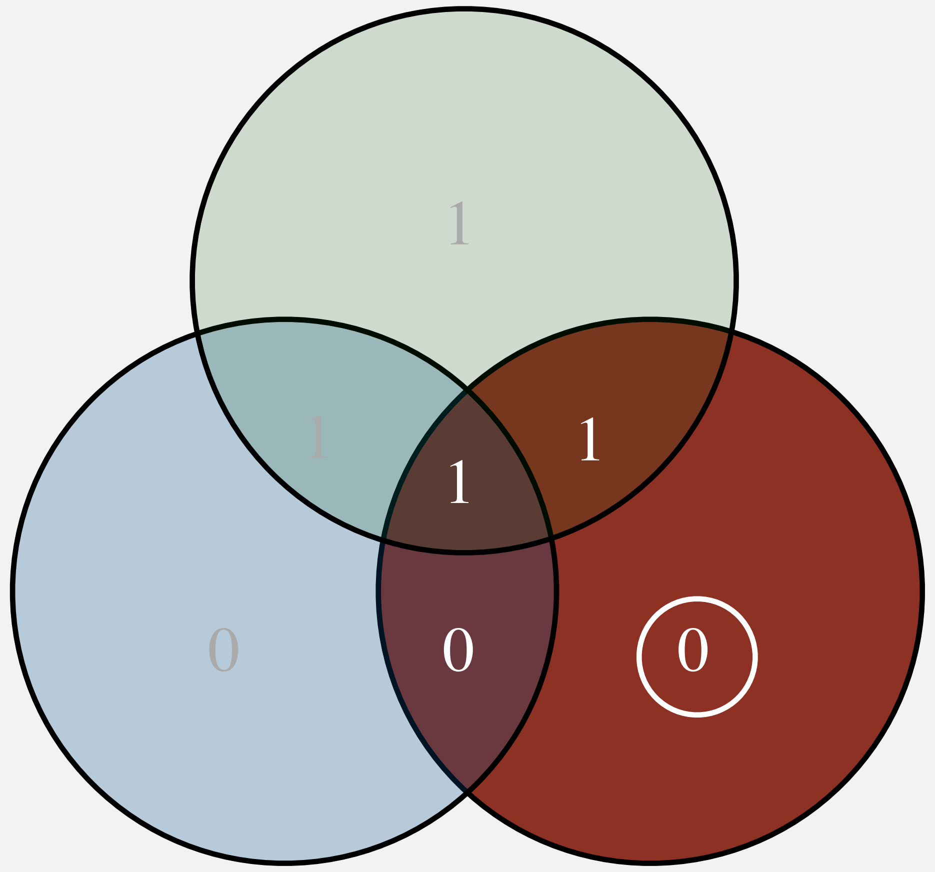
Parity bits. Uniquely chosen so that the sum of bits in each circle is even.



$1 + 1 + 1 + p_1 = \text{even}$



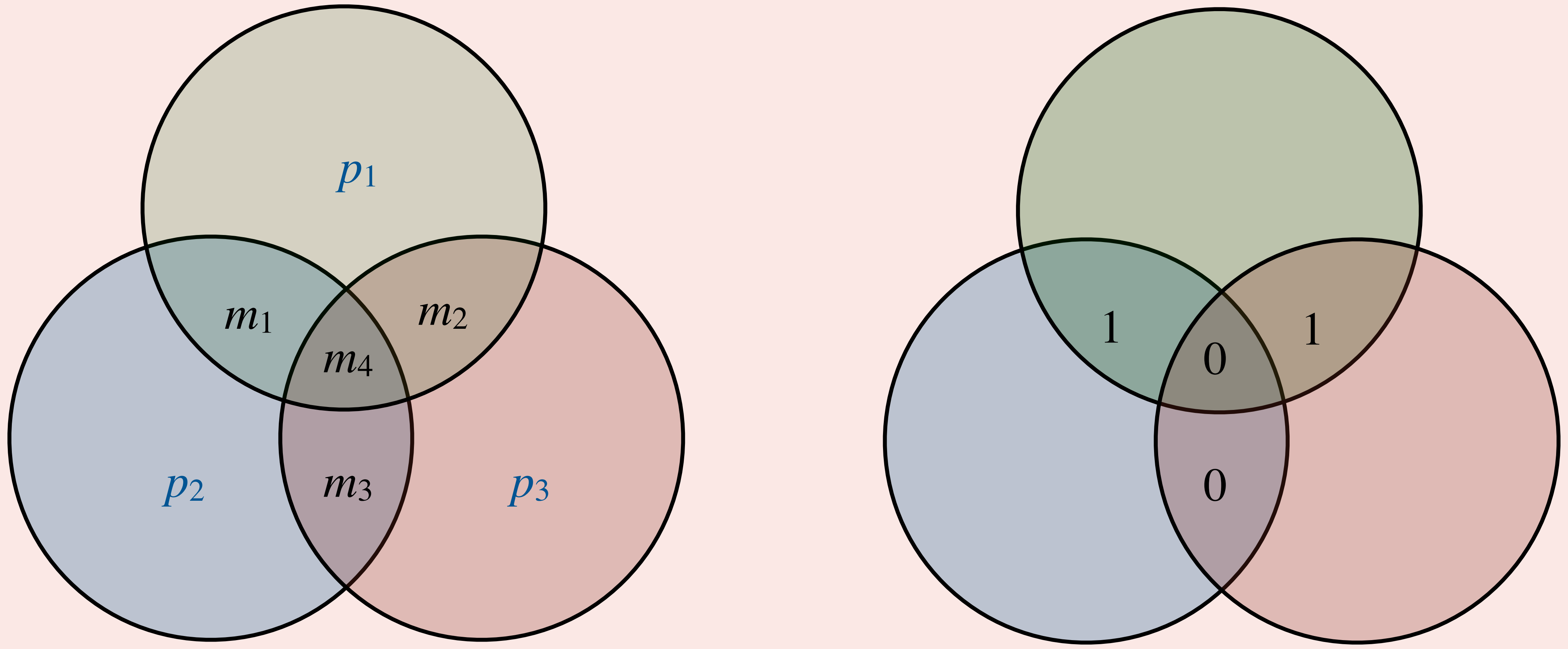
$1 + 1 + 0 + p_2 = \text{even}$



$1 + 1 + 0 + p_3 = \text{even}$

Hamming encoding quiz

QuizSocket.com



Which 7 bits are sent for the message 1 1 0 0 ?

A. 1 1 0 0 0 0 0

B. 1 1 0 0 0 1 0

C. 1 1 0 0 0 1 1

D. 1 1 0 0 1 1 1

Useful trick: the xor function

Hint. Can use the *xor* function to compute parity bits.

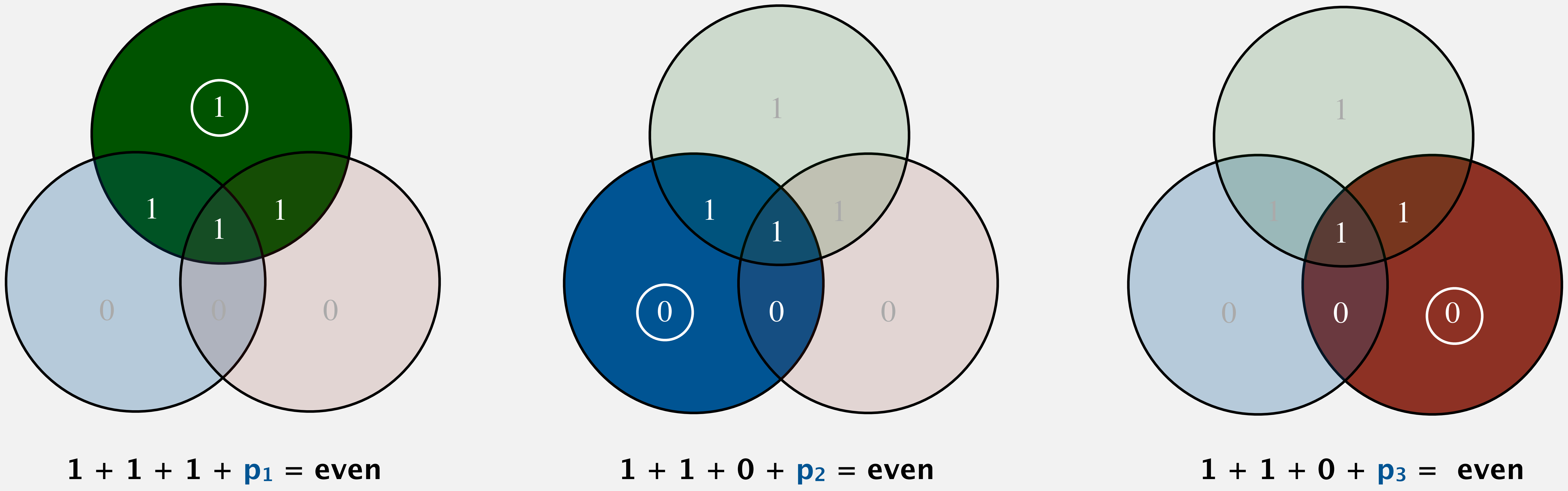
x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

$$p_1 = m_1 \wedge m_2 \wedge m_4$$
$$p_2 = m_1 \wedge m_3 \wedge m_4$$
$$p_3 = m_2 \wedge m_3 \wedge m_4$$

Ex 1. $p_1 = 1 \wedge 1 \wedge 1 = 1.$

Ex 2. $p_2 = 1 \wedge 0 \wedge 1 = 0.$

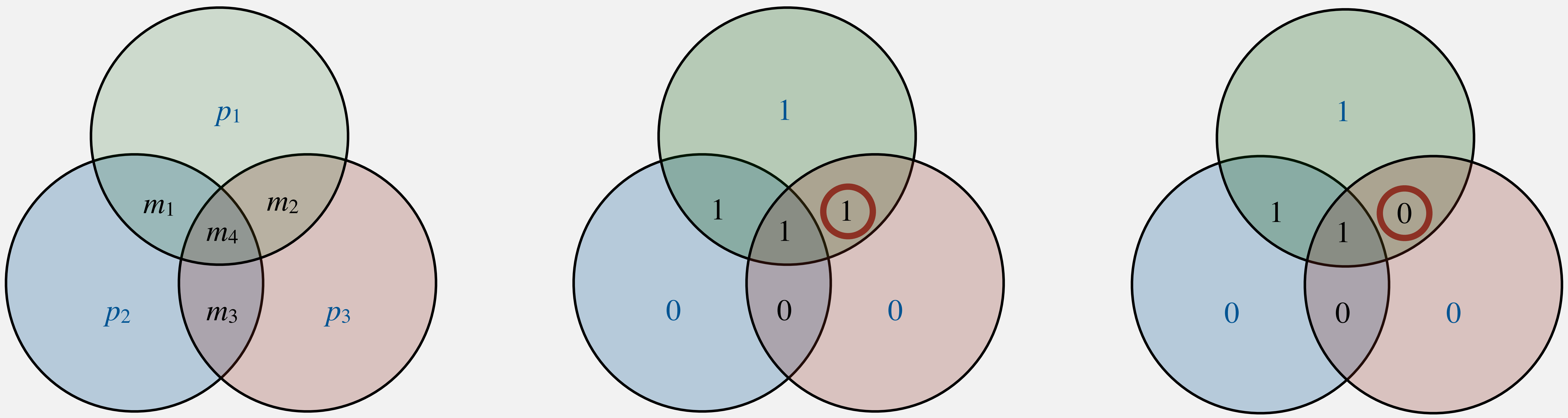
Ex 3. $p_3 = 1 \wedge 0 \wedge 1 = 0.$

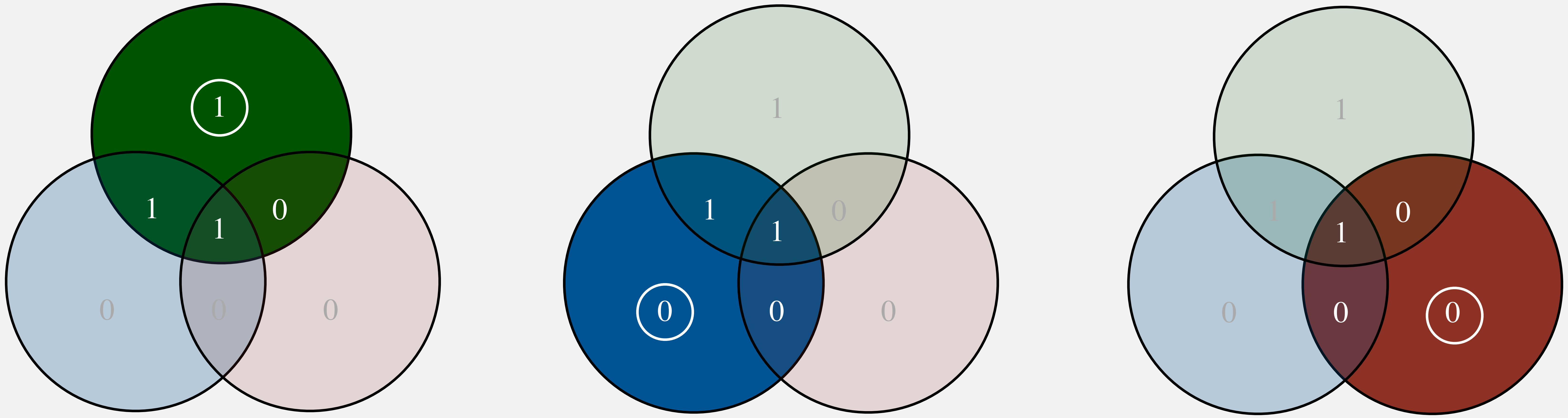


Message bit m_2 flipped

1101100 transmitted

suppose 1**0**01100 received



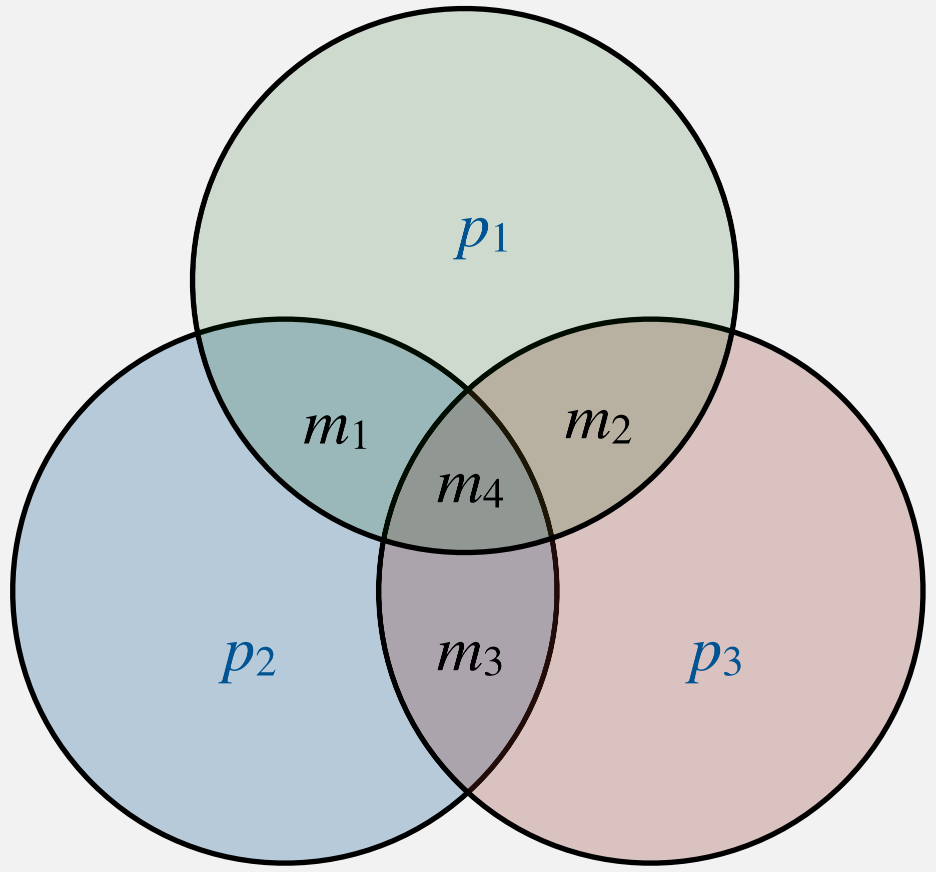


1 + 1 + 0 + 1 = odd
(top fails)

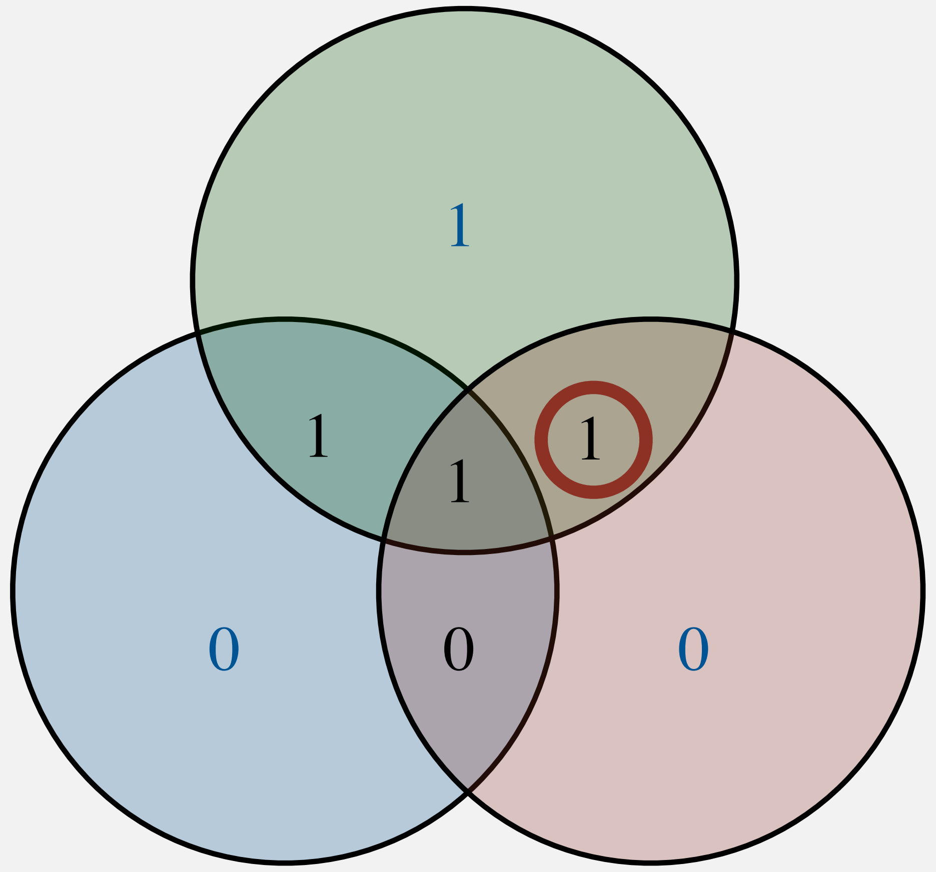
1 + 1 + 0 + 0 = even
(left passes)

1 + 0 + 0 + 0 = odd
(right fails)

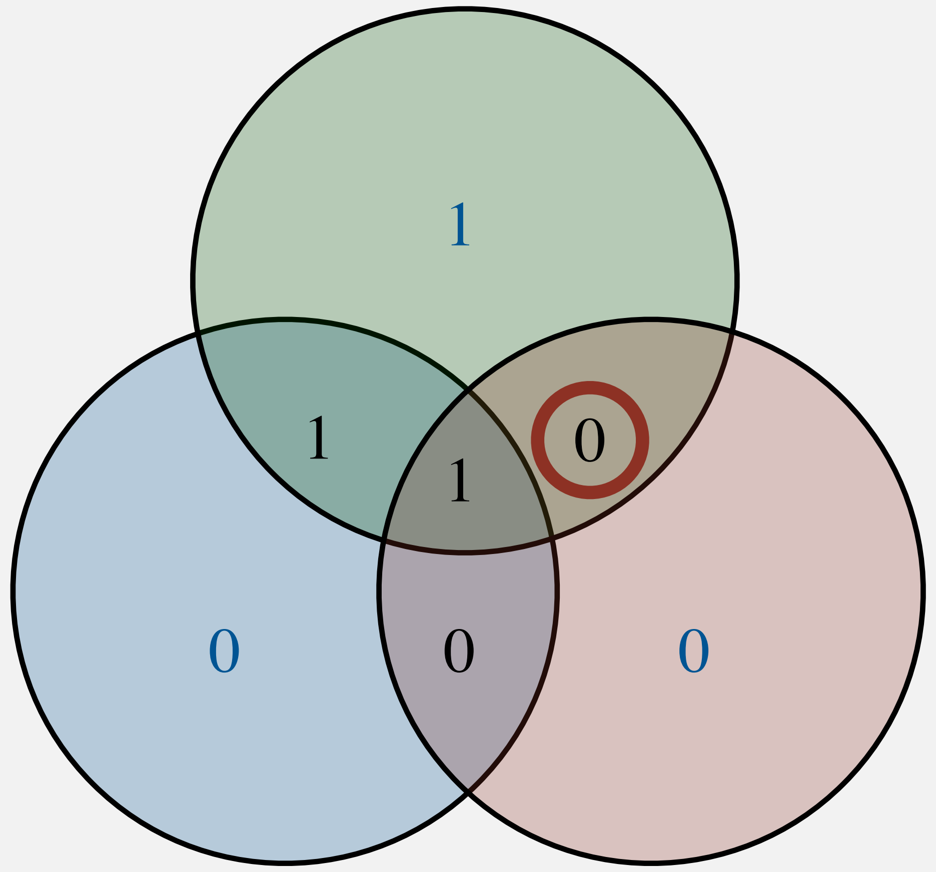
Message bit m_2 flipped

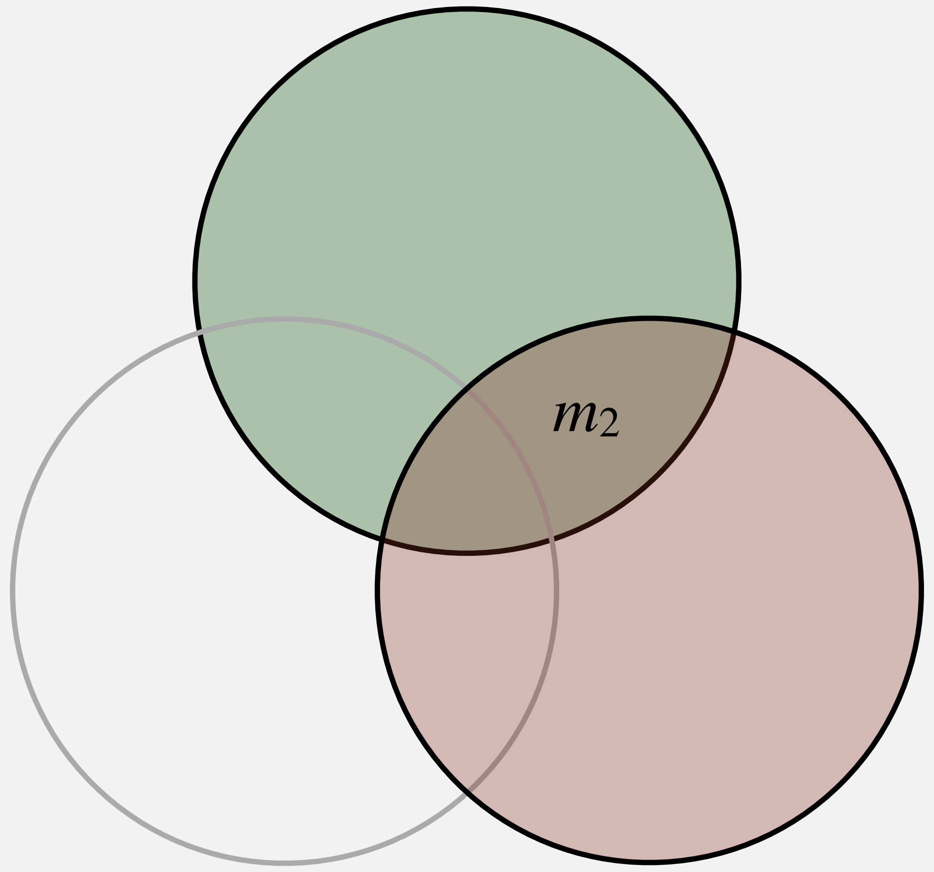


1101100 transmitted



suppose 1001100 received

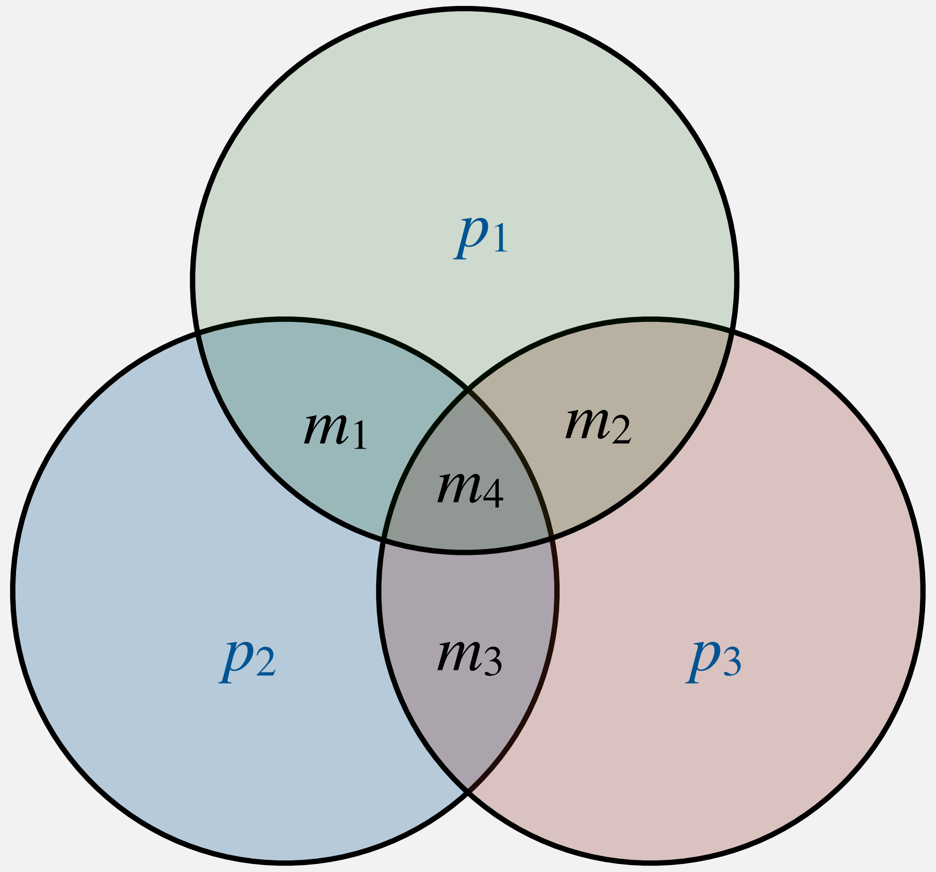




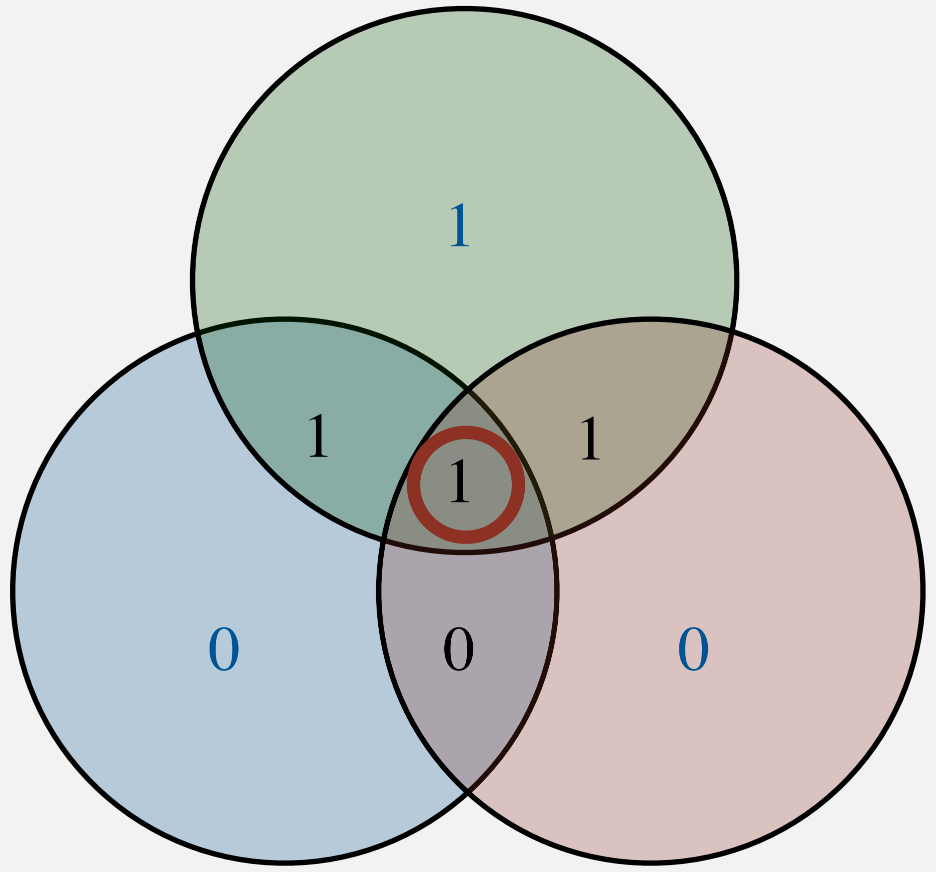
top fails

right fails

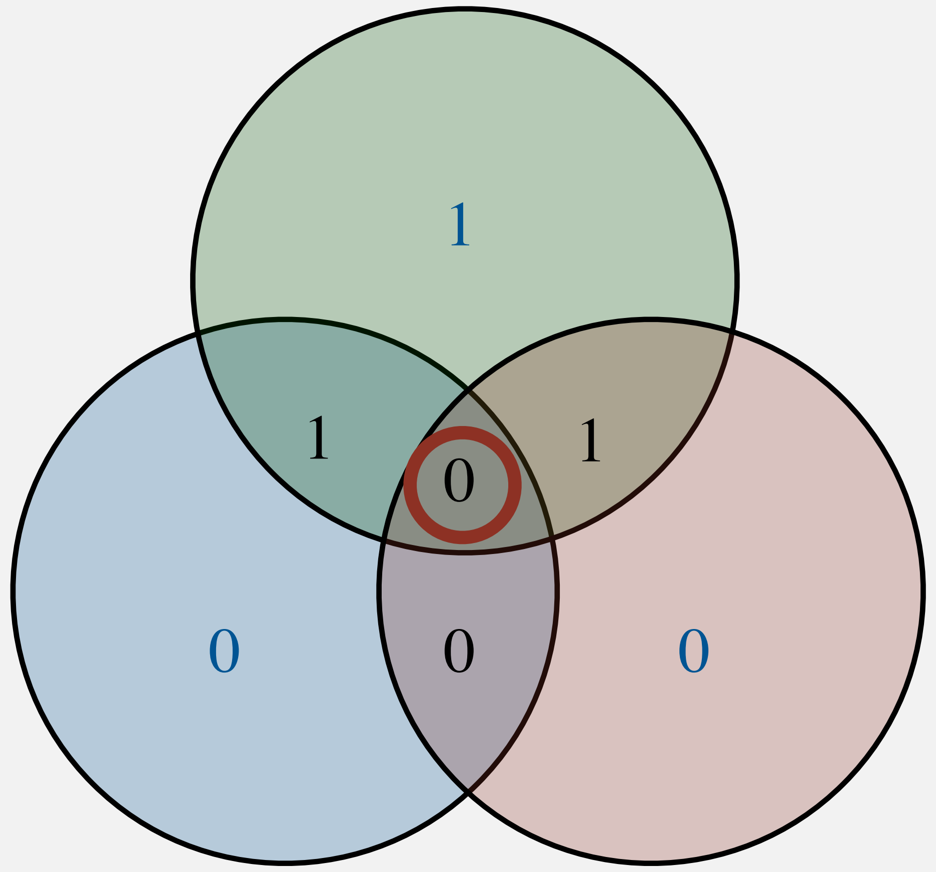
Message bit m_4 flipped

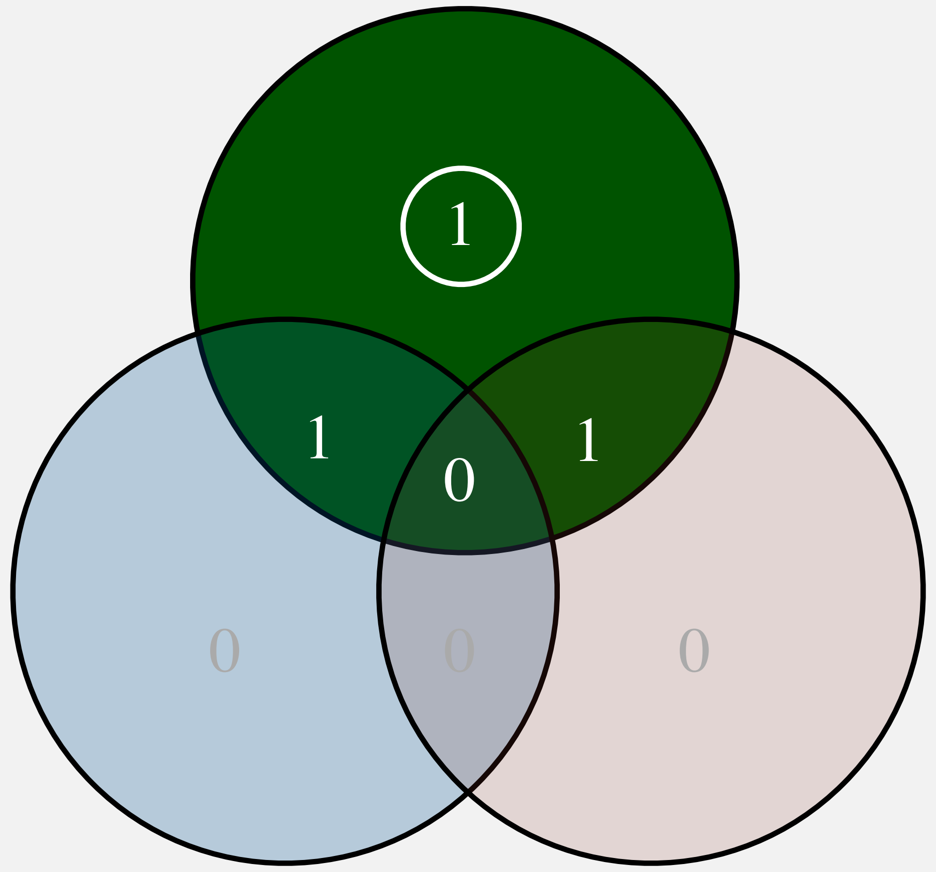


1101100 transmitted

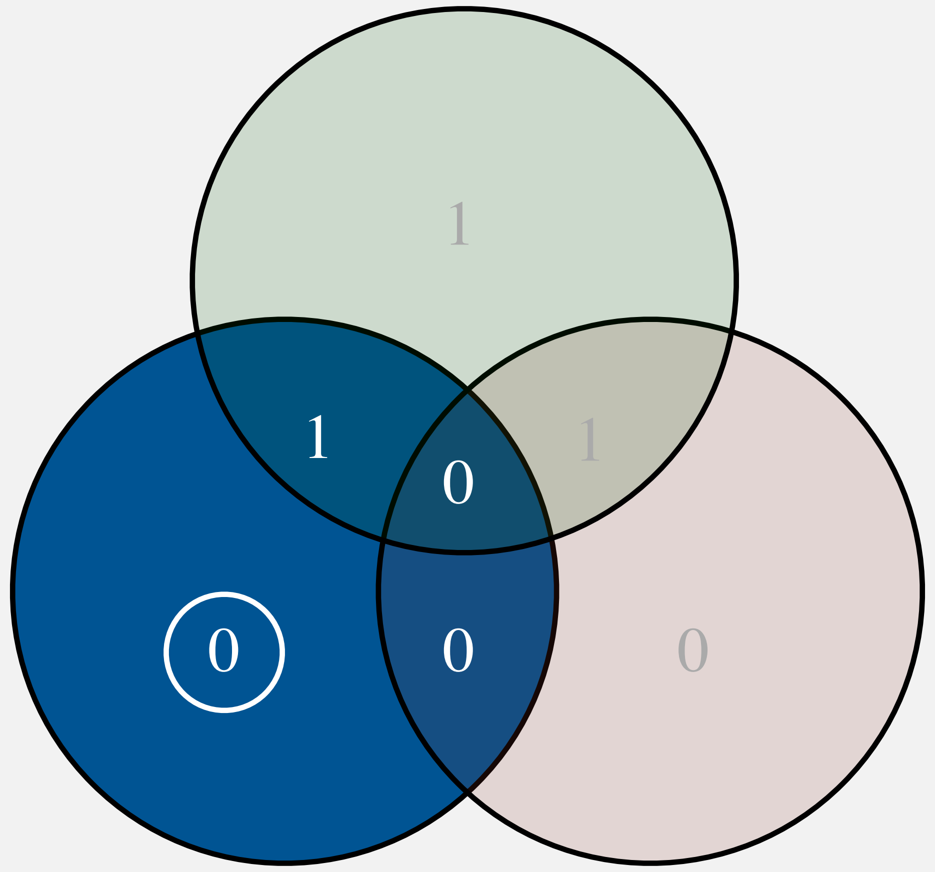


suppose 1100100 received

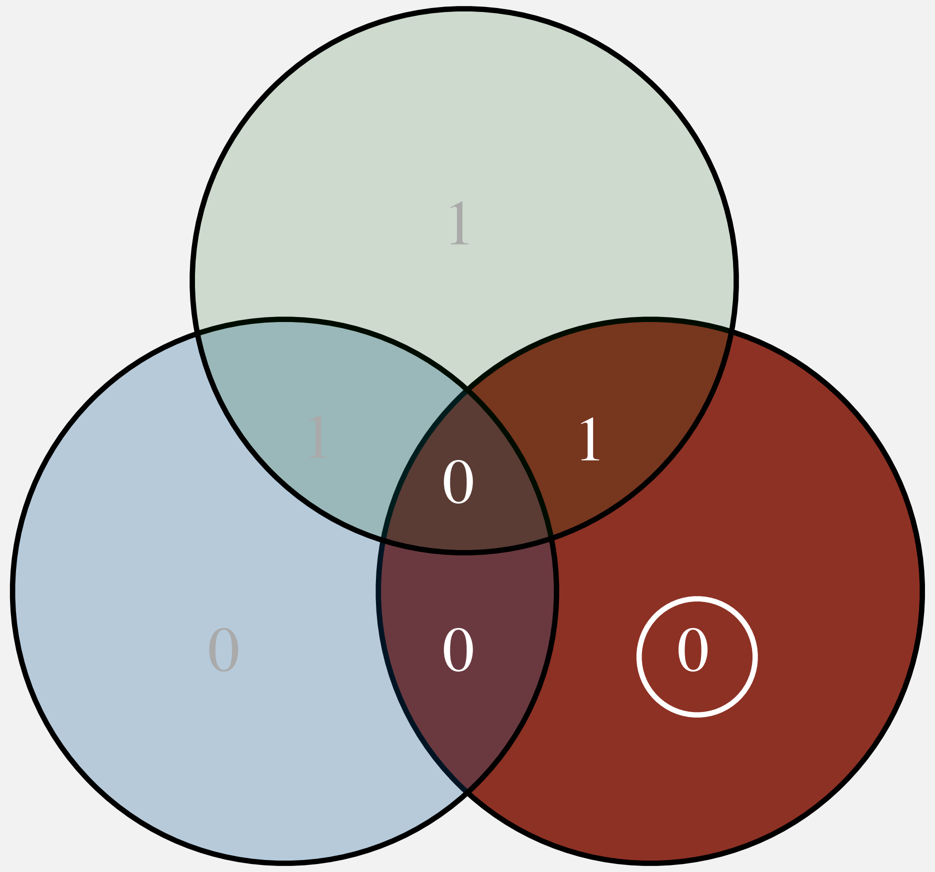




1 + 1 + 0 +1 = odd
(top fails)

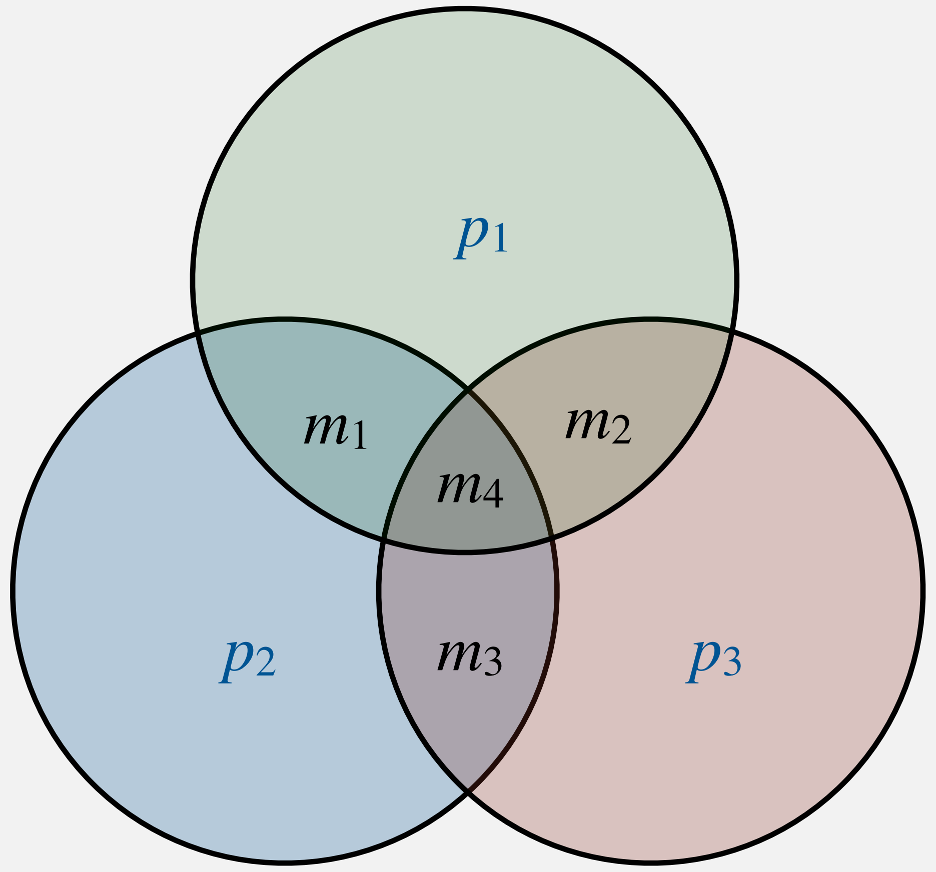


1 + 0 + 0 + 0 = odd
(left fails)

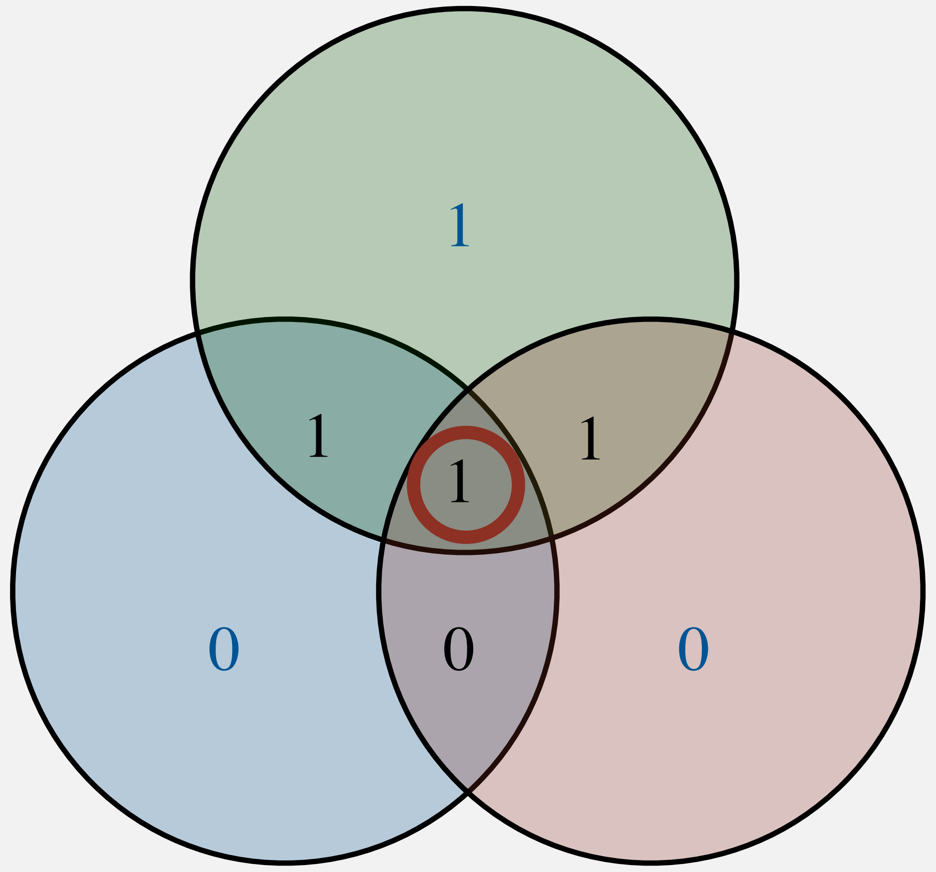


1 + 0 + 0 + 0 = odd
(right fails)

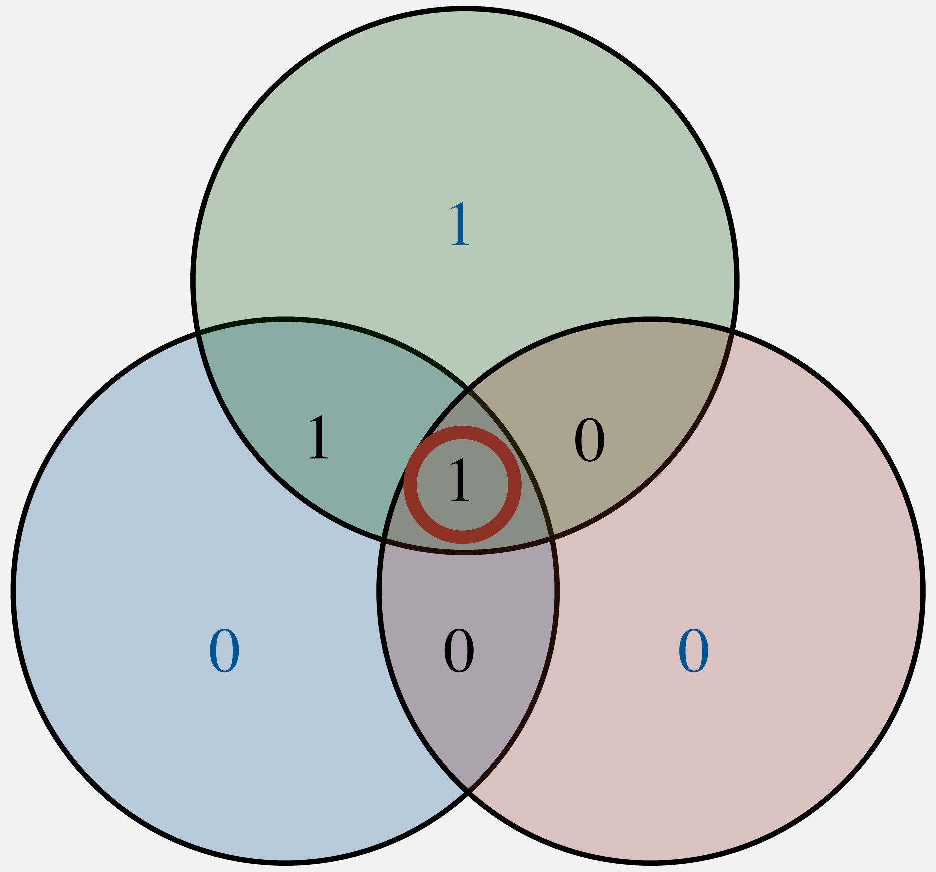
Message bit m_4 flipped

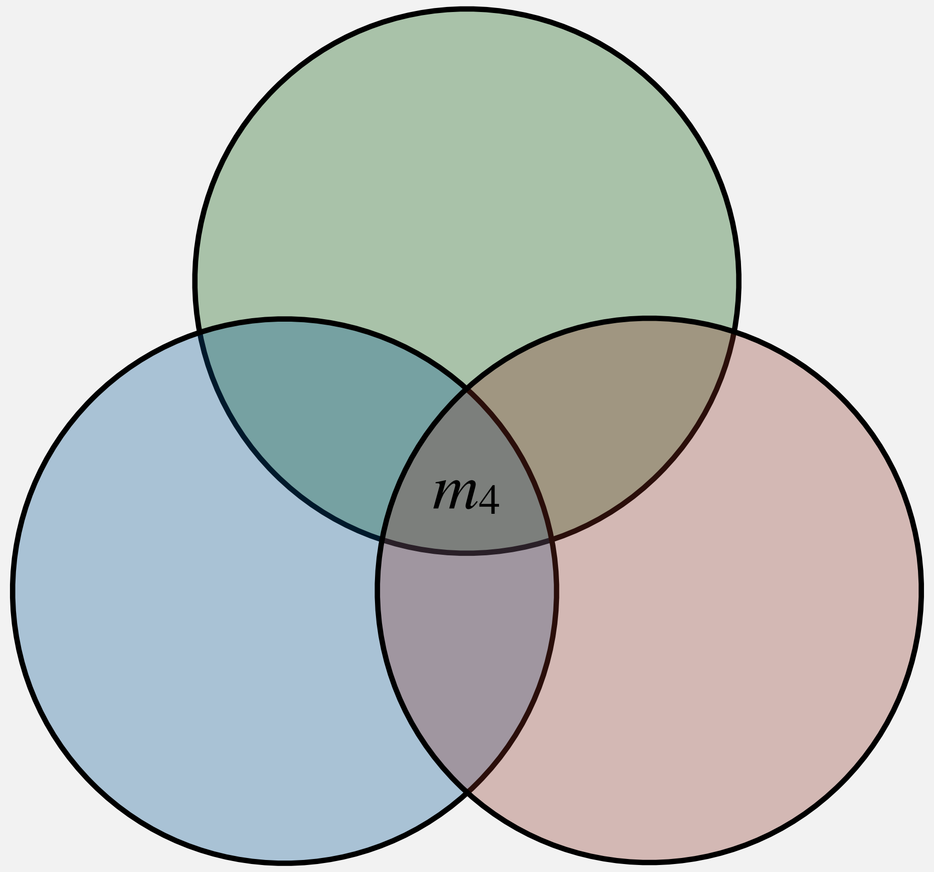


1101100 transmitted



suppose 1001100 received





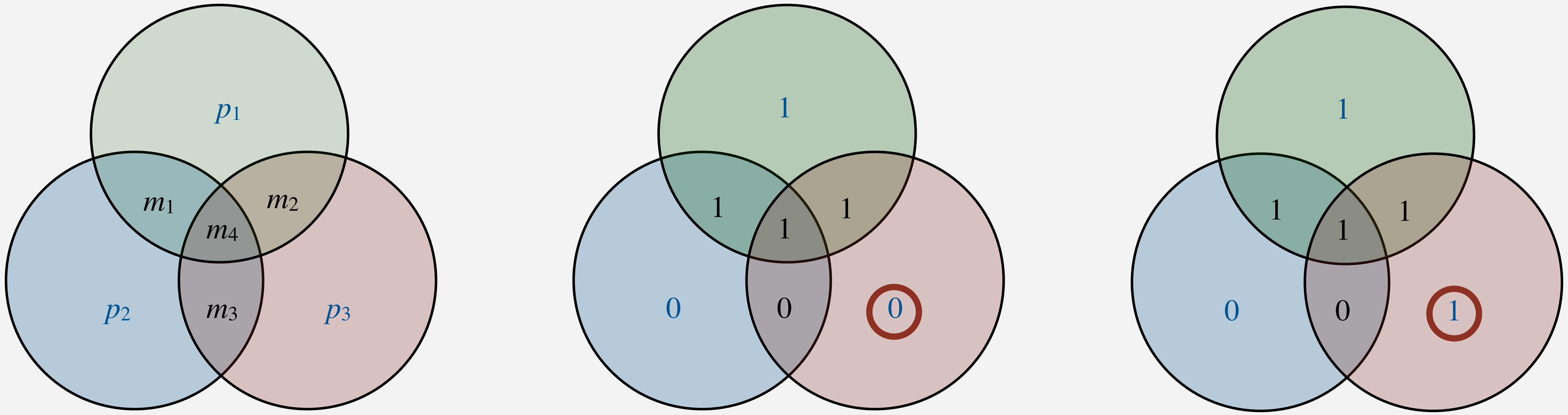
top fails

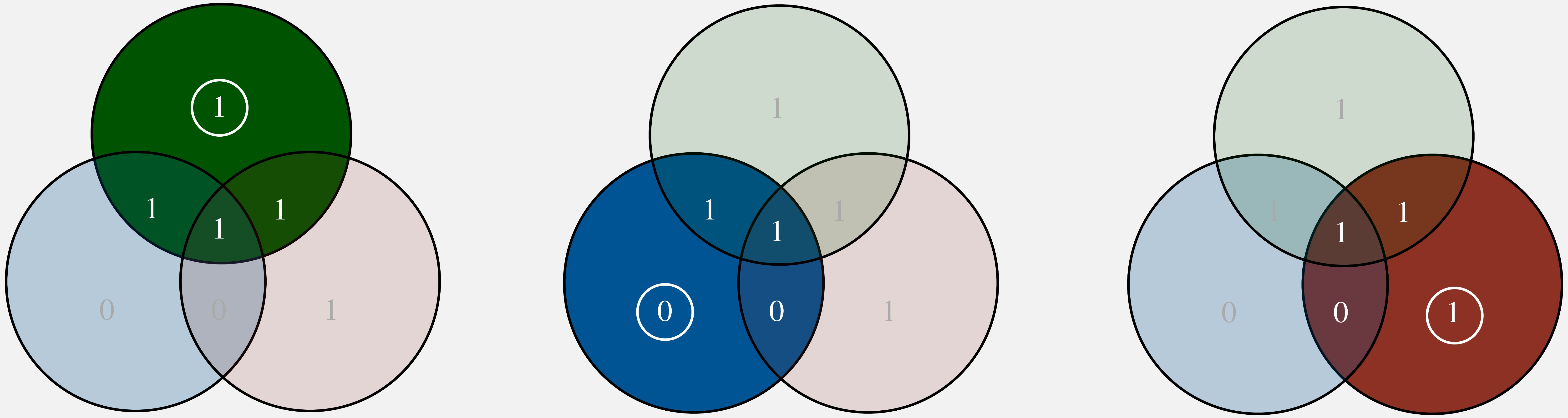
left fails **right fails**

Parity bit p_3 flipped

1101100 transmitted

suppose 110110**1** received



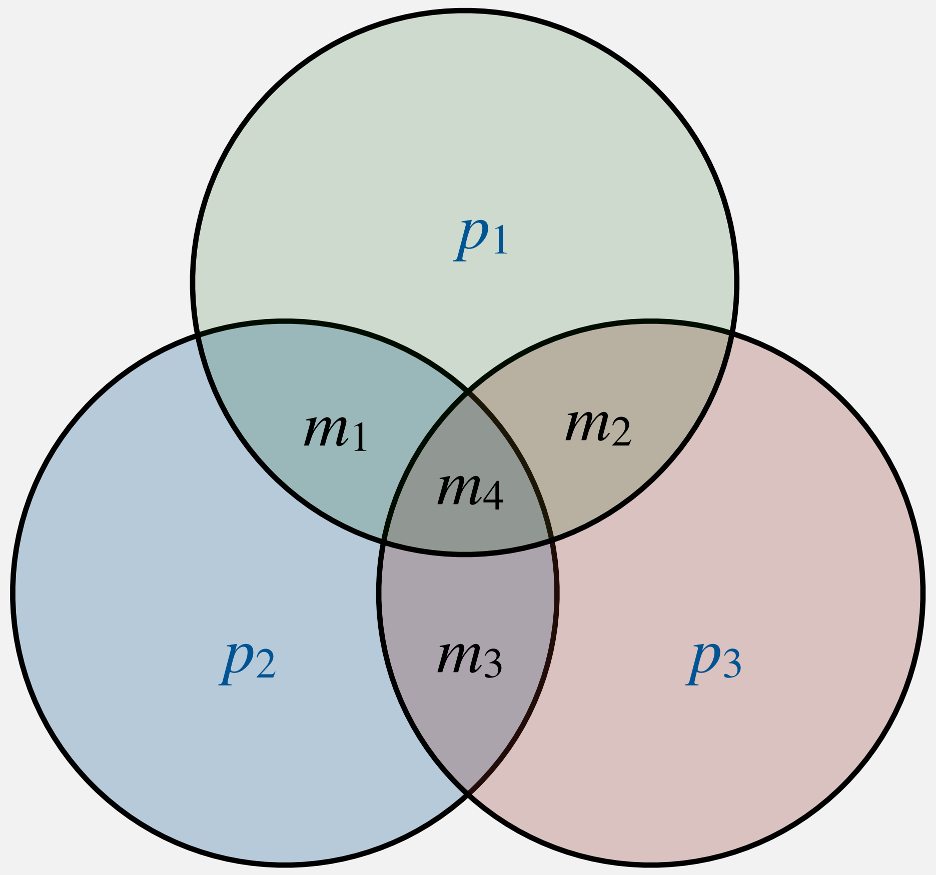


1 + 1 + 1 + 1 = even
(top passes)

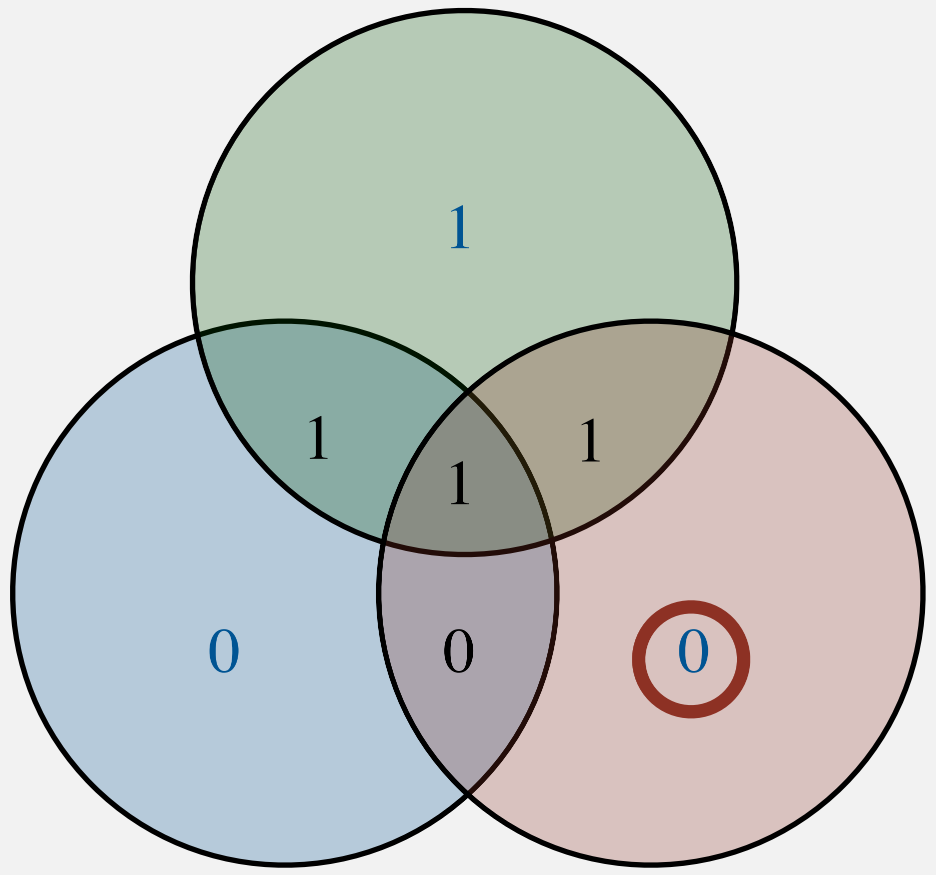
1 + 1 + 0 + 0 = even
(left passes)

1 + 1 + 0 + 1 = odd
(right fails)

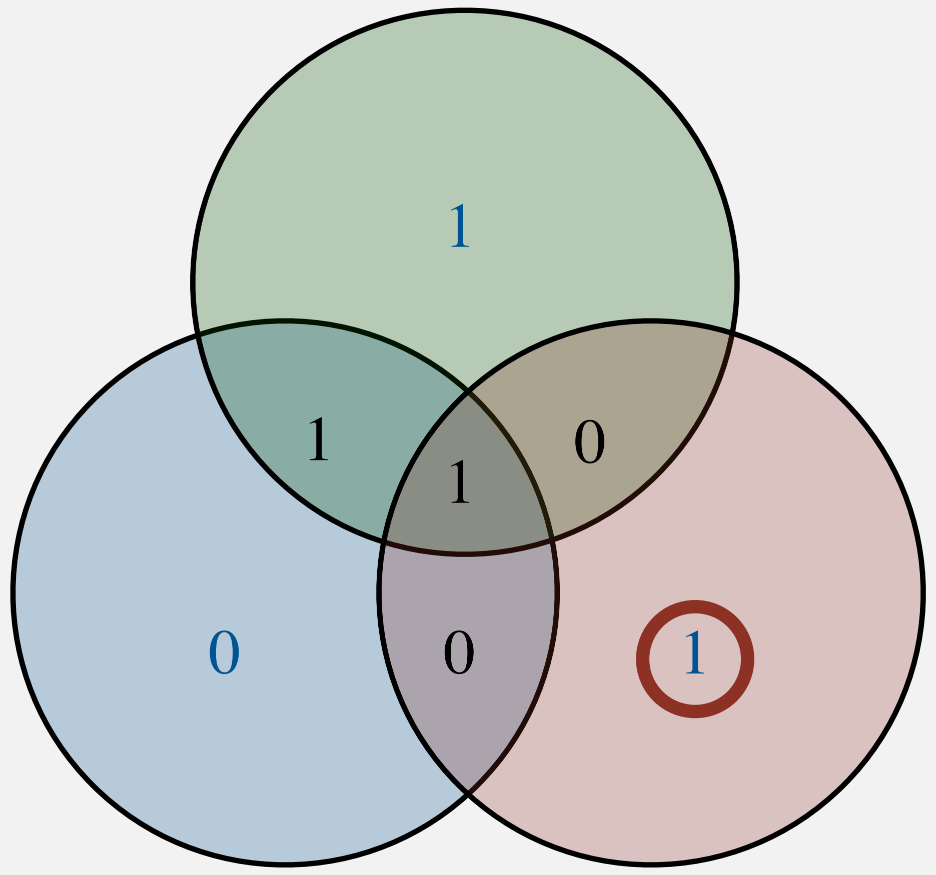
Parity bit p_3 flipped

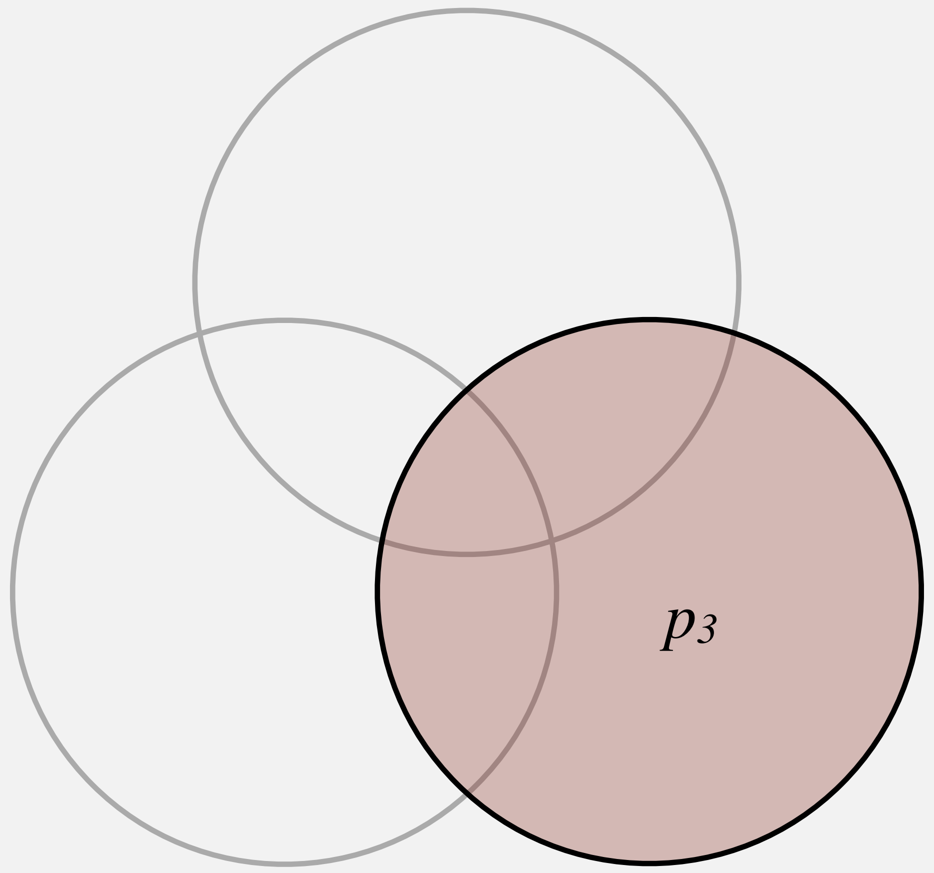


1101100 transmitted



suppose 1101110 received





right fails

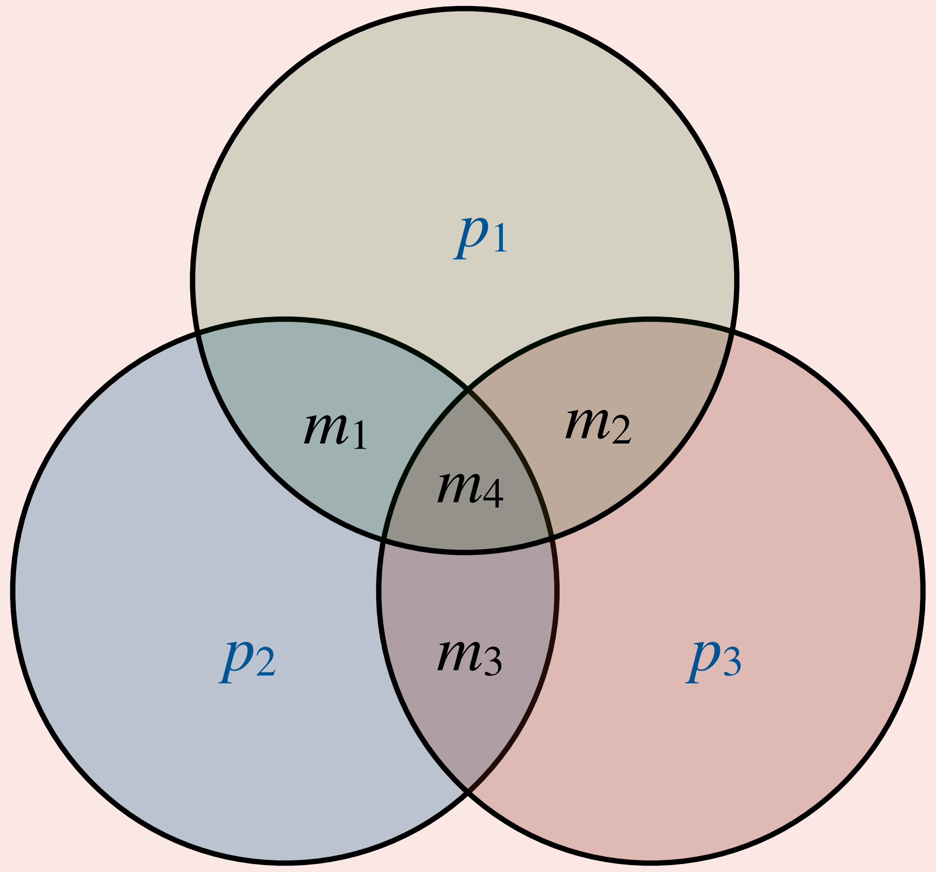
Error correction rule

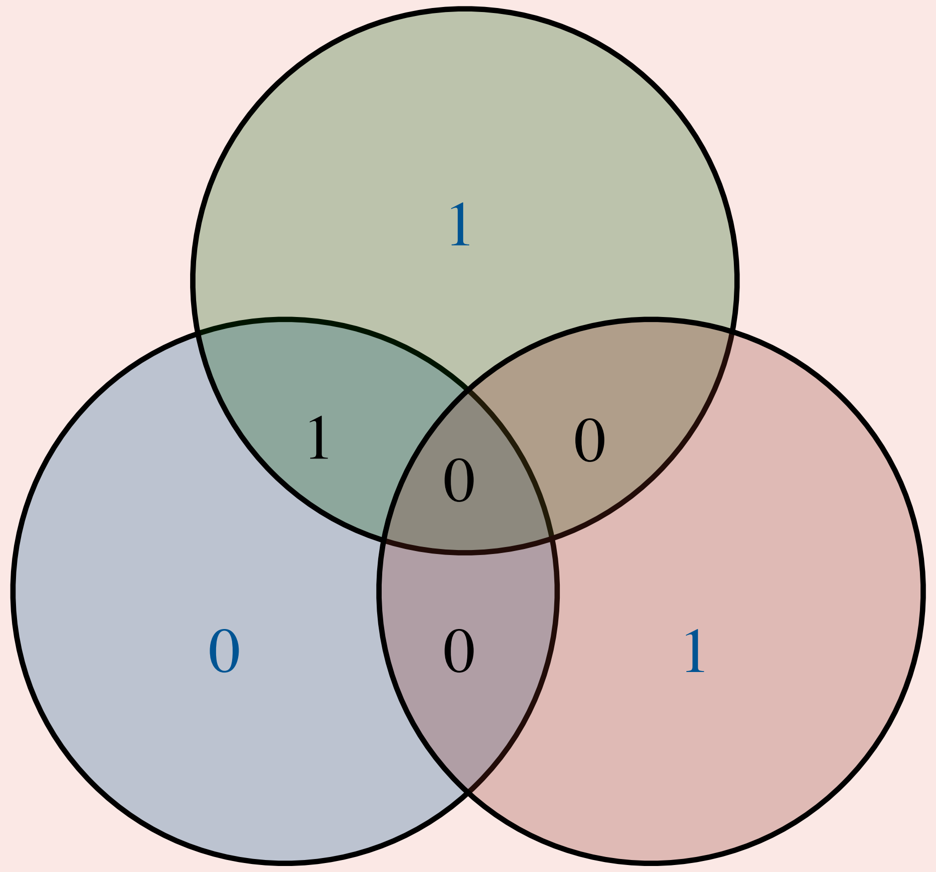
Compute parity bits p_1, p_2, and p_3 and compare against received bits.

- If at most 1 parity check fails, all message bits are correct.
- If all 3 parity checks fail, then message bit m_4 was flipped.
- If only checks p_1 and p_2 fail, then message bit m_1 was flipped.
- If only checks p_1 and p_3 fail, then message bit m_2 was flipped.
- If only checks p_2 and p_3 fail, then message bit m_3 was flipped.

Caveat. 7–4 Hamming code are not designed to detect (or correct) multiple flipped bits.

Hamming decoding quiz





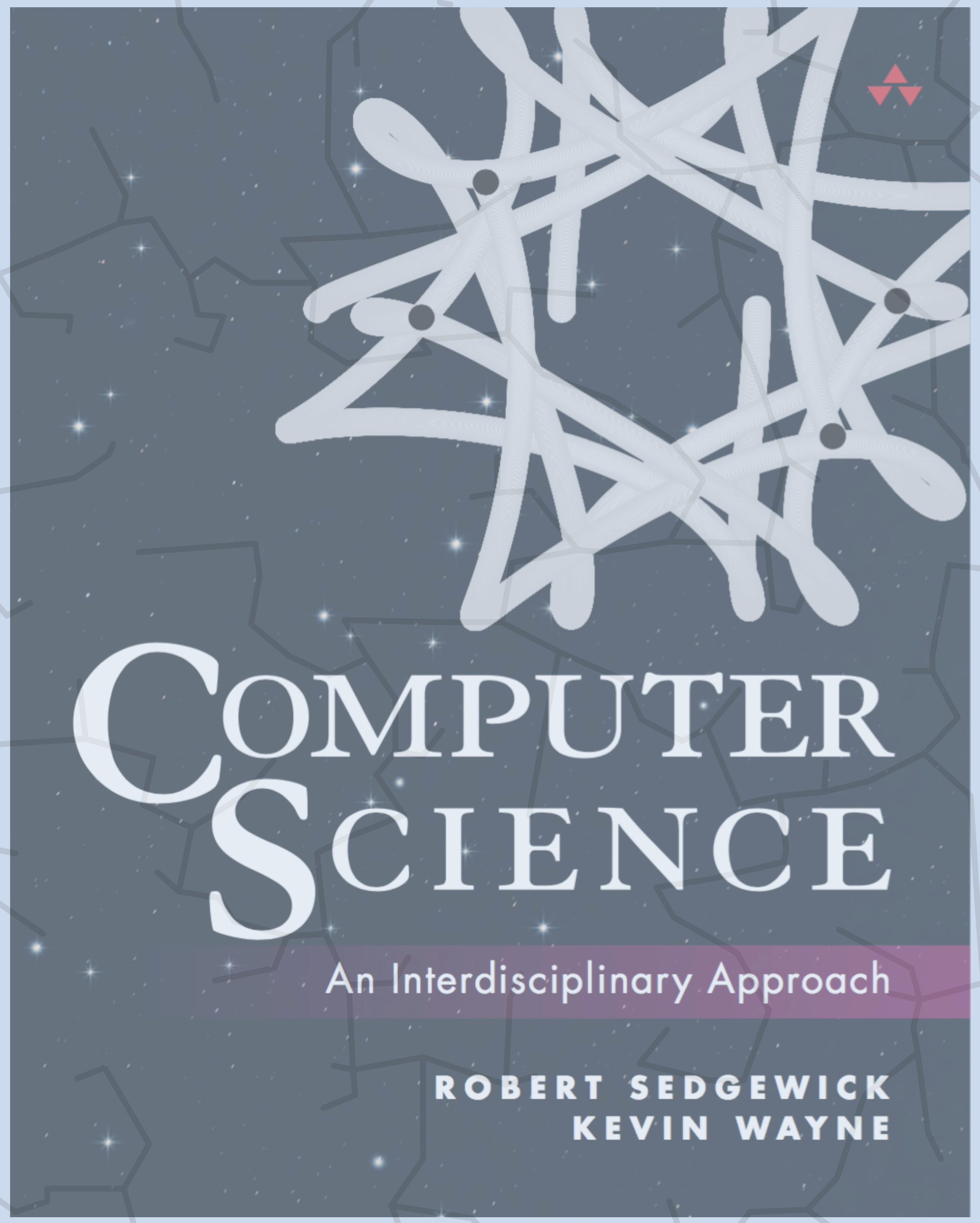
You receive the bits 1 0 0 0 1 0 1 . Which were the original 4 message bits?

A. 0 0 0 0

B. 1 0 0 1

C. 1 0 1 0

D. 1 1 0 0



http://introcs.cs.princeton.edu

Hamming Codes in TOY

- *Hamming codes*
- *TOY simulator*
- *bugs to avoid*

TOY file format

```
% more echo.toy
/*****************************************************************
 *  Name:     Kevin Wayne
 *  NetID:    wayne
 *  Precept:  P00
 *
 *  Description: Reads integers from standard input until 0000;
 *               prints each integer to standard output.
 * ****************************************************************/

10: 81FF   read R[1]
11: C114   if (R[1] == 0) goto 14
12: 91FF   write R[1]
13: C010   goto 10

14: 0000   halt
```

```
while (!StdIn.isEmpty()) {
    a = StdIn.readInt();
    StdOut.println(a);
}
```

- any line not of the form XX : YYYY is ignored by TOY simulator
- one line of TOY code
- TOY pseudo-code
- Java-style comments (optional)
- memory address (in hex) followed by colon
- TOY instruction (in hex)

TOY simulator

Edit file. Use any text editor (such as DrJava).

Not-so-useful feature in DrJava.

- DrJava auto-indents lines.
- Preferences → Miscellaneous → Indent Level = 0.

  [switch back to 4 after this assignment]

Execute. Execute TOY program from command line.

- `TOY.java` must be in same directory as `.toy` files.
- `java-introcs TOY encode.toy < encode3.txt`
- `java-introcs TOY decode.toy < decode5.txt`

```
% more encode3.txt
0001 0001 0000 0001
0001 0001 0001 0000
0001 0001 0001 0001
FFFF
```

4 bits to encode

for simplicity, each bit stored as 16-bit TOY word

end of file convention

7 bits to encode

```
% more decode3.txt
0001 0000 0000 0001 0001 0000 0000
0000 0001 0001 0000 0000 0000 0000
0001 0001 0001 0001 0001 0001 0000
FFFF
```

Visual X-TOY simulator

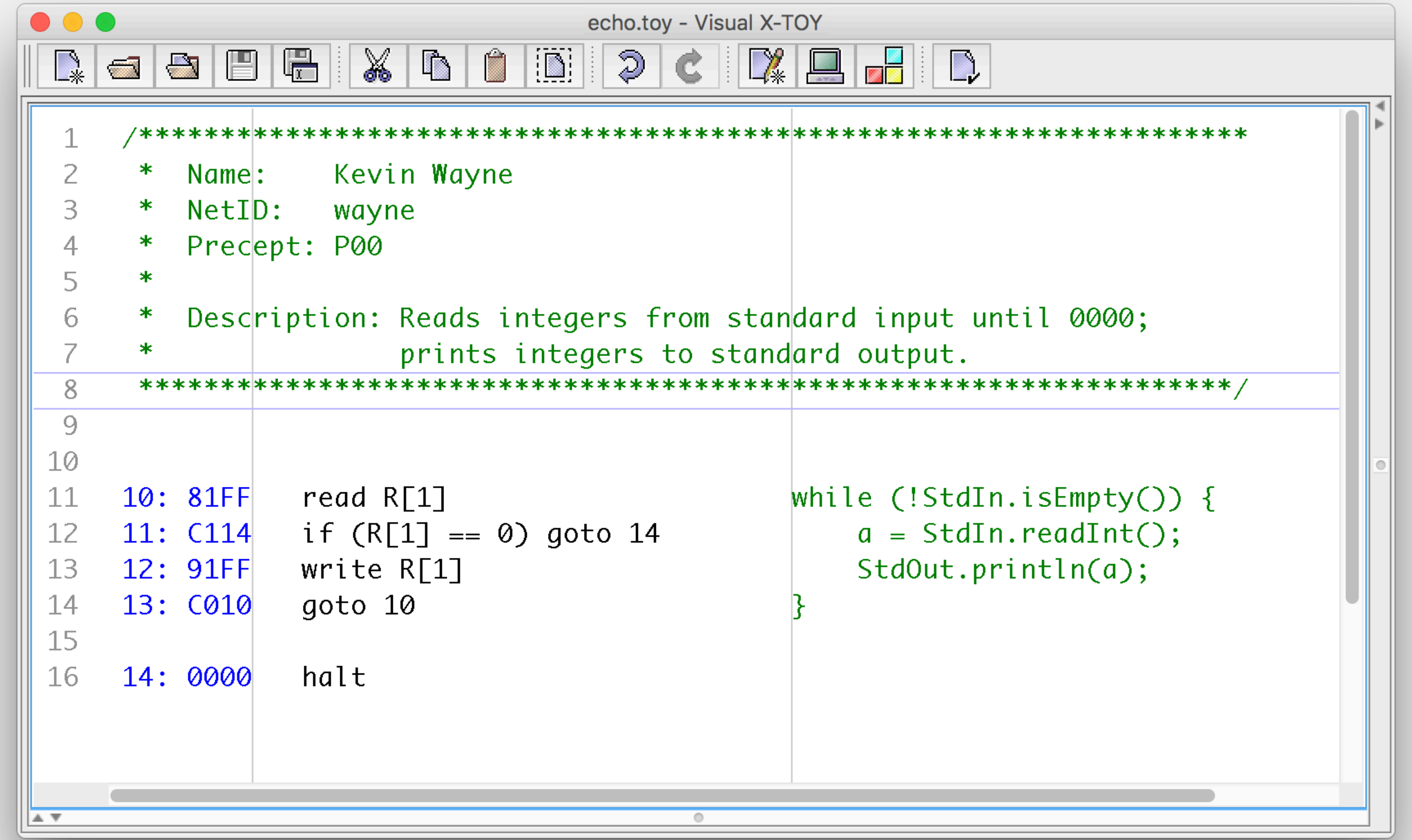
Edit mode. Write your TOY program.

Debug mode. Execute your TOY program.

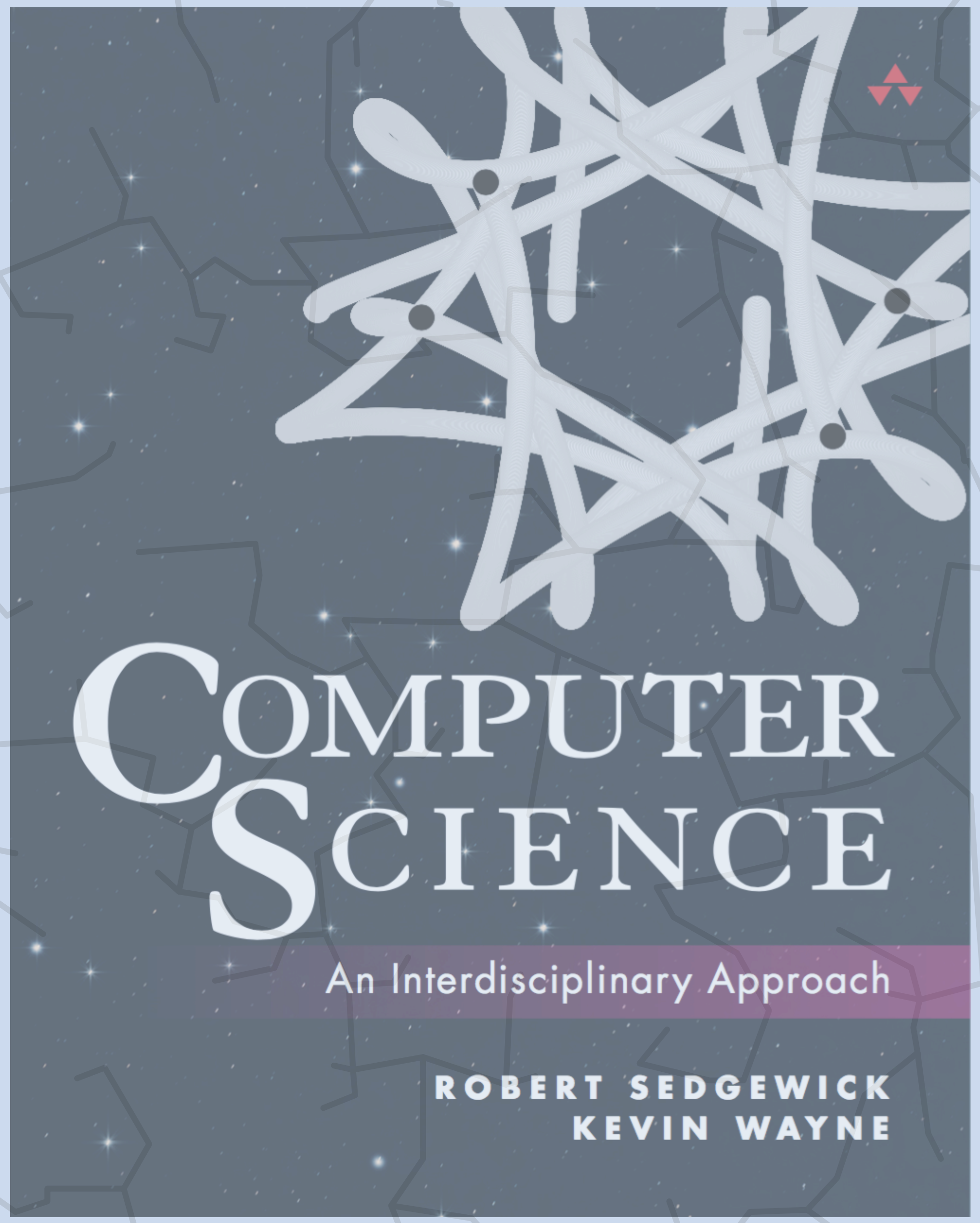
Simulation mode. For historical context.

Useful features.

- Syntax highlighting.
- Automatically generates TOY pseudo-code.
- Tools → Check Syntax.
- Mode → Load File to Stdin.



written by Brian Tsang '04



http://introcs.cs.princeton.edu

HAMMING CODES IN TOY

- *Hamming codes*
- *TOY simulator*
- *bugs to avoid*

Tips to avoid common bugs

- Start your TOY code at line 10.
- Check that each line of TOY code has format `XX:YYYY`.
- Remember that “everything” is in hex (line 1A follows 19).
- Make sure TOY code and pseudo-code match.
- Document the purpose of each register (and don’t reuse).
- Use care when inserting a line of code:

  might need to update jump statement if line to goto changes.
- Repeatedly read 4- or 7-bits from standard input until FFFF.