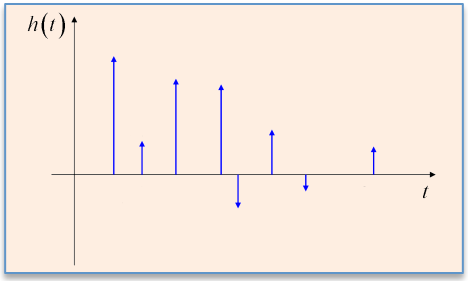
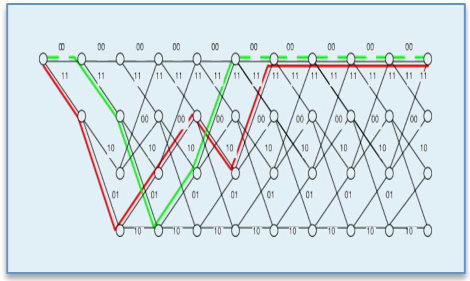
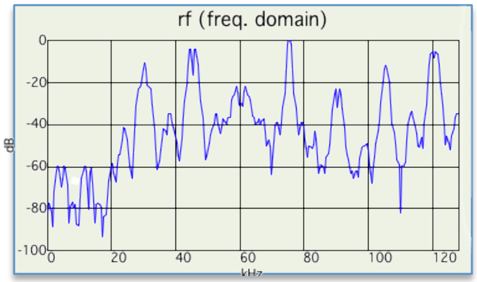
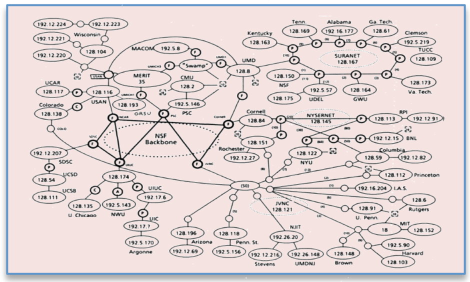








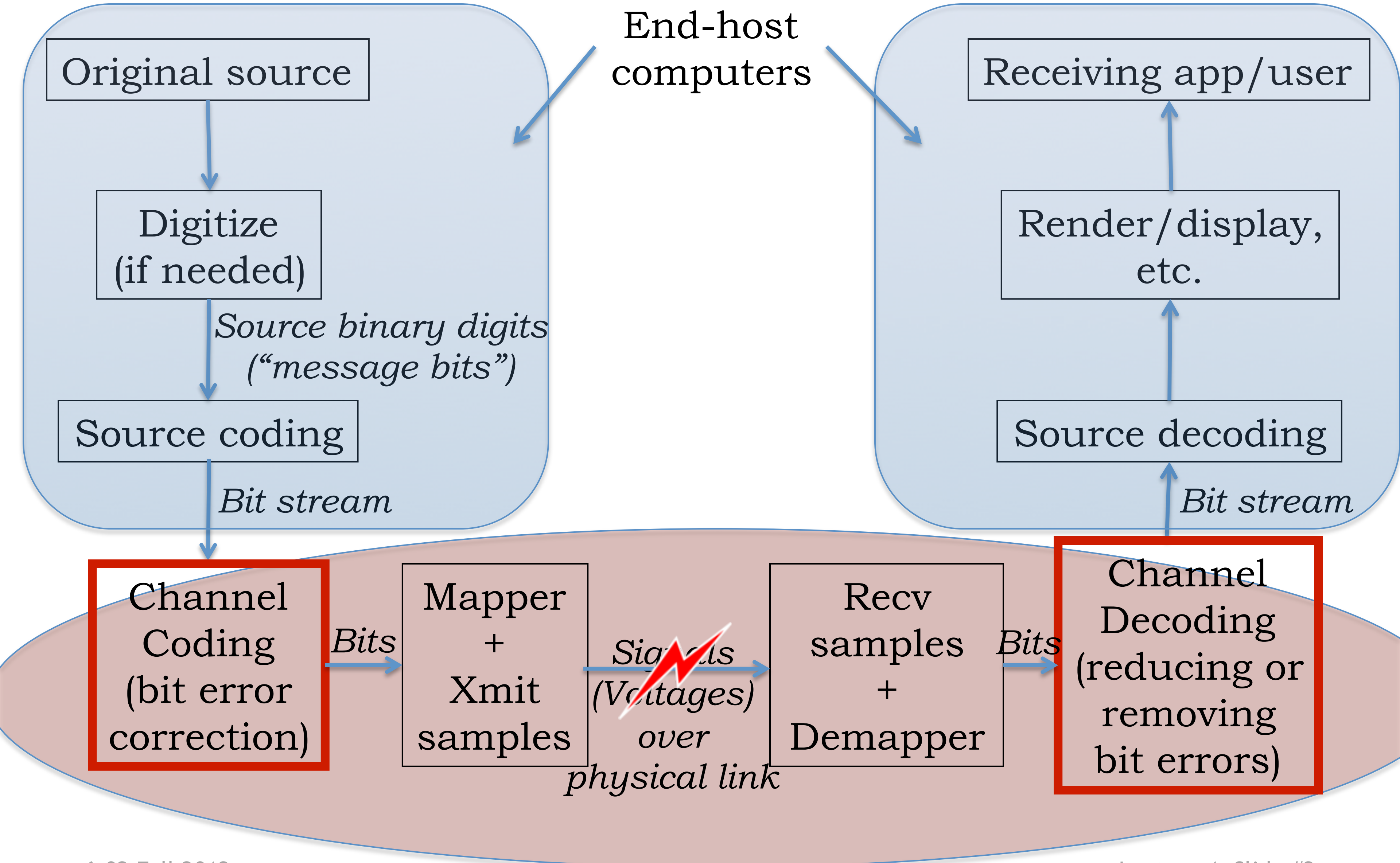
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2012
Lecture #4

- Linear block codes
  - Rectangular codes
  - Hamming codes

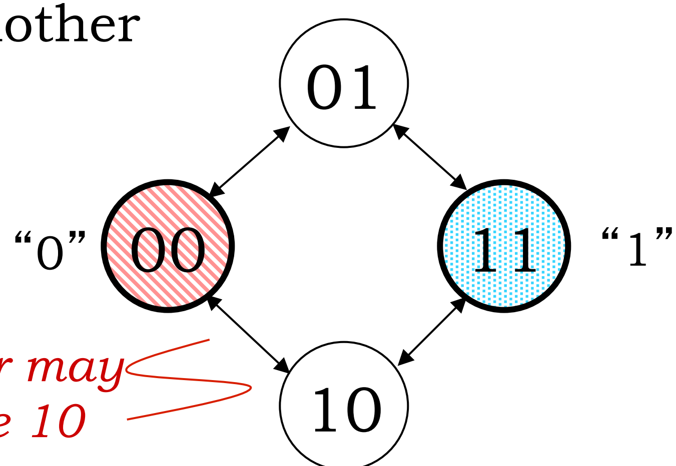
Single Link Communication Model



Embedding for Structural Separation

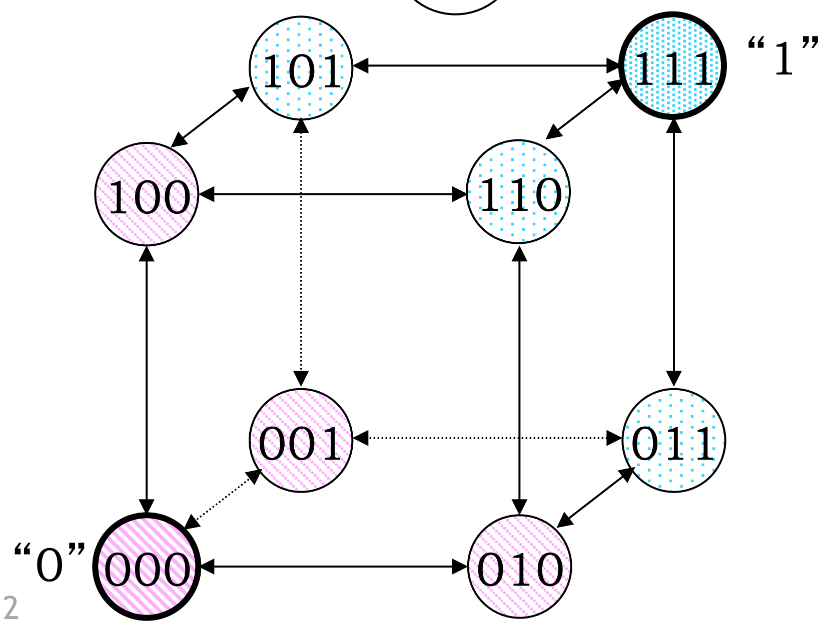
Encode so that the codewords are "far enough" from each other

Likely error patterns shouldn't transform one codeword to another



single-bit error may cause 00 to be 10 (or 01)

Code: nodes chosen in hypercube + mapping of message bits to nodes



If we choose 2^k out of 2^n nodes, it means we can map all k-bit message strings in a space of n-bit *codewords*. The ***code rate*** is **k/n**.

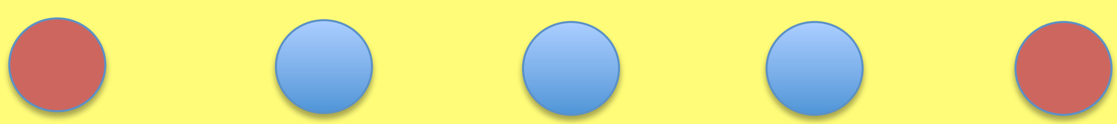
Minimum Hamming Distance of Code vs. Detection & Correction Capabilities

If d is the minimum Hamming distance between codewords, we can:

- detect **all** patterns of up to t bit errors
  if and only if $d \geq t+1$

- correct **all** patterns of up to t bit errors
  if and only if $d \geq 2t+1$

- detect **all** patterns of up to t_D bit errors
  while correcting all patterns of t_C ($<t_D$) errors
  if and only if $d \geq t_C+t_D+1$

e.g.:  $d=4$, $t_C=1$, $t_D=2$

Linear Block Codes

Block code: ***k*** message bits encoded to **n** code bits i.e., each of $\mathbf{2}^{k}$ messages encoded into a unique **n-bit** codeword via a *linear transformation.*

Key property: Sum of any two codewords is *also* a codeword → necessary and sufficient for code to be linear.

(n,k) code has rate **k/n**.
Sometime written as **(n,k,d)**, where **d** is the minimum Hamming Distance of the code.

Generator Matrix of Linear Block Code

Linear transformation:

$$C=D.G$$

C is an n-element row vector containing the codeword

D is a k-element row vector containing the message

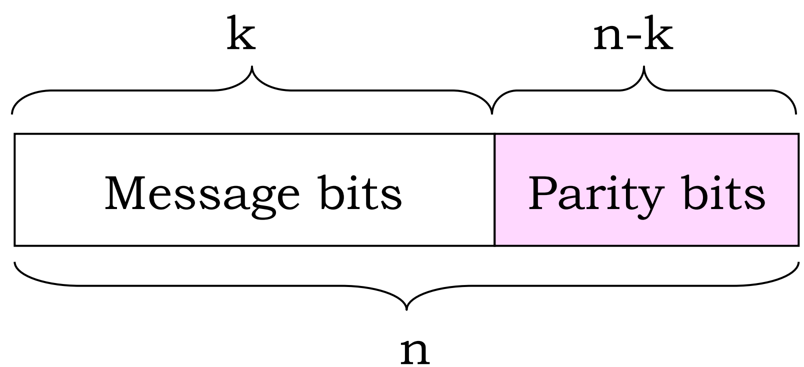
G is the kxn *generator matrix*

Each codeword bit is a specified linear combination of message bits.

Each codeword is a linear combination of rows of G.

(n,k) Systematic Linear Block Codes

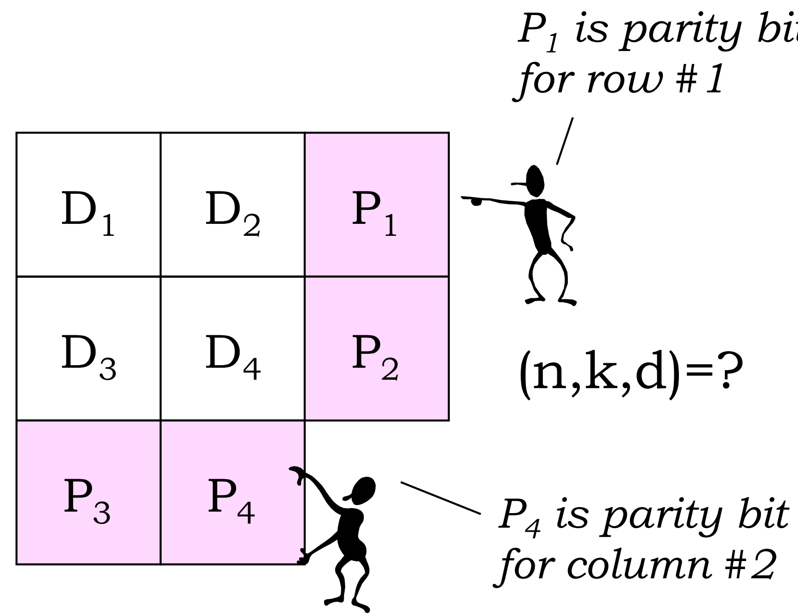
- Split data into *k*-bit blocks
- Add (*n*-*k*) parity bits to each block using (*n*-*k*) linear equations, making each block *n* bits long



- Every linear code can be represented by an equivalent systematic form --- ordering is not significant, direct inclusion of k message bits in n-bit codeword is.
- Corresponds to using invertible transformations on rows and permutations on columns of G to get
- G = [I | A] --- identity matrix in the first k columns

Example: Rectangular Parity Codes

Idea: start with rectangular array of data bits, add parity checks for each row and column. Single-bit error in data will show up as parity errors in a particular row and column, pinpointing the bit that has the error.



(n,k,d)=?

0 1 1 1 1 0 1 0	0 1 1 1 0 0 1 0	0 1 1 1 1 1 1 0
Parity for each row and column is correct ⇒ no errors	Parity check fails for row #2 and column #2 ⇒ bit D_4 is incorrect	Parity check only fails for row #2 ⇒ bit P_2 is incorrect

Rectangular Code Corrects Single Errors

Claim: The min HD of the rectangular code with ***r*** rows and ***c*** columns is **3**. Hence, it is a single error correction (SEC) code.

Code rate = $rc / (rc + r + c)$.

If we add an overall parity bit P, we get a (rc+r+c+1, rc, 4) code

Improves error detection but not correction capability

D_1	D_2	D_3	D_4	P_1
D_5	D_6	D_7	D_8	P_2
D_9	D_{10}	D_{11}	D_{12}	P_3
P_4	P_5	P_6	P_7	P

Proof: Three cases.

(1) Msgs with HD 1 → differ in 1 row and 1 col parity

(2) Msgs with HD 2 → differ in either 2 rows OR 2 cols or both → HD ≥ 4

(3) Msgs with HD 3 or more → HD ≥ 4

Matrix Notation

Task: given k-bit message, compute n-bit codeword. We can use standard matrix arithmetic (modulo 2) to do the job. For example, here's how we would describe the (9,4,4) rectangular code that includes an overall parity bit.

$$D_{1xk} \cdot G_{kxn} = C_{1xn}$$

$$\begin{bmatrix} D_1 & D_2 & D_3 & D_4 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} D_1 & D_2 & D_3 & D_4 & P_1 & P_2 & P_3 & P_4 & P_5 \end{bmatrix}$$

1×k message vector

k×n generator matrix

1×n code word vector

The generator matrix, $G_{kxn} = \begin{bmatrix} I_{k\times k} & \Big| & A_{k\times(n-k)} \end{bmatrix}$

Decoding Rectangular Parity Codes

Receiver gets possibly corrupted word, ***w***.

Calculates all the parity bits from the data bits.

If no parity errors, return ***rc*** bits of data.

Single row or column parity bit error → ***rc*** data bits are fine, return them

If parity of row ***x*** and parity of column ***y*** are in error, then the data bit in the **(*x*,*y*)** position is wrong; flip it and return the ***rc*** data bits

All other parity errors are *uncorrectable*. Return the data as-is, flag an "uncorrectable error"

Let's do some rectangular parity decoding

D1	D2	P1
D3	D4	P2
P3	P4	

Received codewords

1	0	1
0	1	0
0	1	

1. Decoder action: ________________

0	0	0
1	1	1
1	1	

2. Decoder action: ________________

0	0	1
0	1	0
0	0	

3. Decoder action: ________________

How Many Parity Bits Do Really We Need?

- We have **n-k** parity bits, which collectively can represent $\mathbf{2^{n-k}}$ possibilities
- For **single-bit error** correction, parity bits need to represent two sets of cases:
  - Case 1: No error has occurred (1 possibility)
  - Case 2: Exactly one of the code word bits has an error (**n** possibilities, not **k**)

- So we need $\mathbf{n+1 \le 2^{n-k}}$

$$n \le 2^{n-k} - 1$$

- Rectangular codes satisfy this with big margin --- inefficient

Hamming Codes

- Hamming codes correct single errors with the minimum number of parity bits:

$$n = 2^{n-k} - 1$$

- (7,4,3)
- (15,11,3)
- $(2^m - 1, 2^m - 1 - m, 3)$

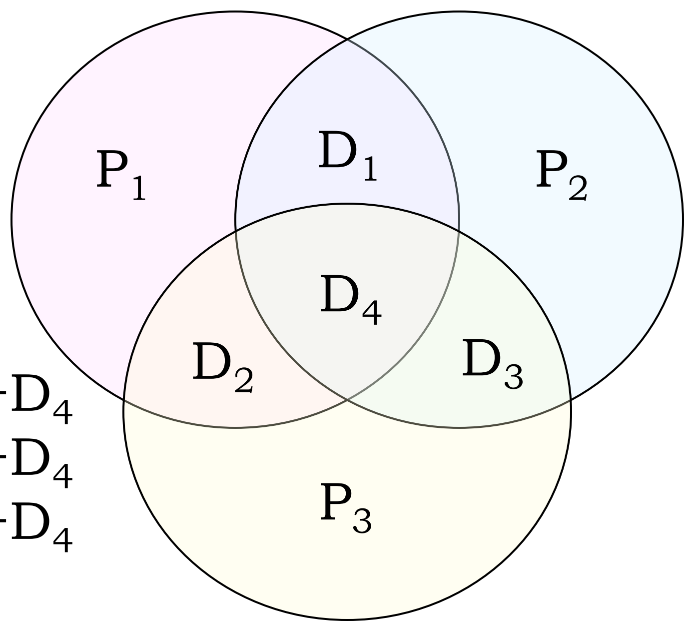
- --- "perfect codes" (but not best!)

Towards More Efficient Codes: (7,4,3) Hamming Code Example

- Use minimum number of parity bits, each covering a subset of the data bits.
- No two message bits belong to exactly the same subsets, so a <u>single-bit error</u> will generate a unique set of parity check errors.

Modulo-2 addition, aka XOR

$P_1 = D_1 + D_2 + D_4$
$P_2 = D_1 + D_3 + D_4$
$P_3 = D_2 + D_3 + D_4$



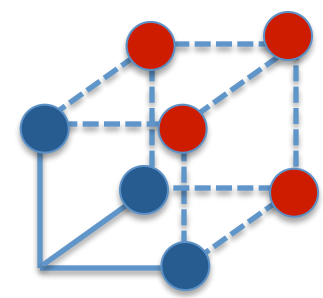
Suppose we check the parity and discover that P1 and P3 indicate an error?
bit D2 must have flipped

What if only P2 indicates an error?
P2 itself had the error!

Logic Behind Hamming Code Construction

- Idea: Use parity bits to cover each axis of the binary vector space
  - That way, all message bits will be covered with a **unique** combination of parity bits

Index	1	2	3	4	5	6	7
Binary index	001	010	011	100	101	110	111
(7,4) code	**P1**	**P2**	**D1**	**P3**	**D2**	**D3**	**D4**



$P_1 = D_1+D_2+D_4$
$P_2 = D_1+D_3+D_4$
$P_3 = D_2+D_3+D_4$

P_1 with binary index 00**1** covers

D_1 with binary index 01**1**
D_2 with binary index 10**1**
D_4 with binary index 11**1**

Syndrome Decoding: Idea

- After receiving the possibly corrupted message (use ' to indicate possibly erroneous symbol), compute a **syndrome** bit (E_i) for each parity bit

$$E_1 = D'_1 + D'_2 + D'_4 + P'_1$$
$$E_2 = D'_1 + D'_3 + D'_4 + P'_2$$
$$E_3 = D'_2 + D'_3 + D'_4 + P'_3$$

$$0 = D_1+D_2+D_4+P_1$$
$$0 = D_1+D_3+D_4+P_2$$
$$0 = D_2+D_3+D_4+P_3$$

- If all the E_i are zero: no errors
- Otherwise use the particular combination of the E_i to figure out correction

Index	1	2	3	4	5	6	7
Binary index	001	010	011	100	101	110	111
(7,4) code	P1	P2	D1	P3	D2	D3	D4

$E_3E_2E_1$	Corrective Action
000	no errors
001	p_1 has an error, flip to correct
010	p_2 has an error, flip to correct
011	d_1 has an error, flip to correct
100	p_3 has an error, flip to correct
101	d_2 has an error, flip to correct
110	d_3 has an error, flip to correct
111	d_4 has an error, flip to correct

Constraints for more than single-bit errors

Code parity constraint inequality for **single-bit** errors

$$1 + n \le 2^{n-k}$$

Write-out the inequality for **t-bit** errors

Elementary Combinatorics

- Given n objects, in how many ways can we choose m of them?

If the ordering of the m selected objects matters, then

$$n(n-1)(n-2) \ldots (n-m+1) = n!/(n-m)!$$

If the ordering of the m selected objects doesn't matter, then the above expression is too large by a factor m!, so

"n choose m" = $$\binom{n}{m} = \frac{n!}{(n-m)!m!}$$

Error-Correcting Codes occur in many other contexts too

- e.g., ISBN numbers for books,
  0-691-12418-3

(Luenberger's *Information Science*)

- $1D_1 + 2D_2 + 3D_3 + \ldots + 10D_{10} = 0 \bmod 11$

Detects single-digit errors, and transpositions