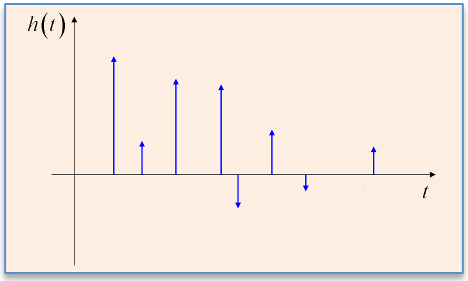
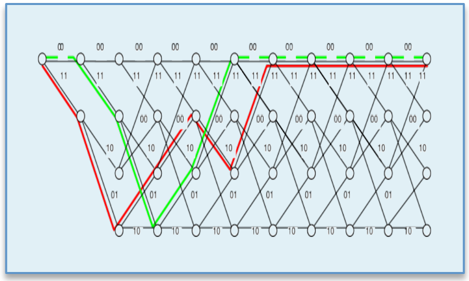
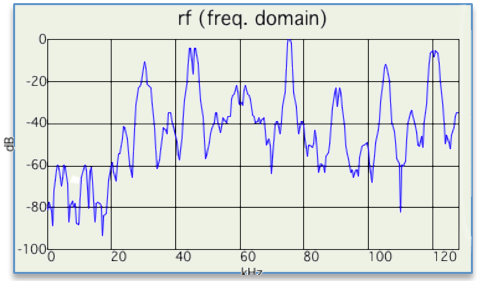
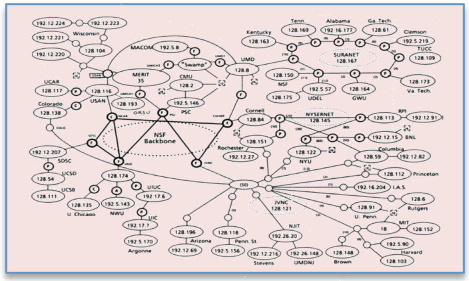




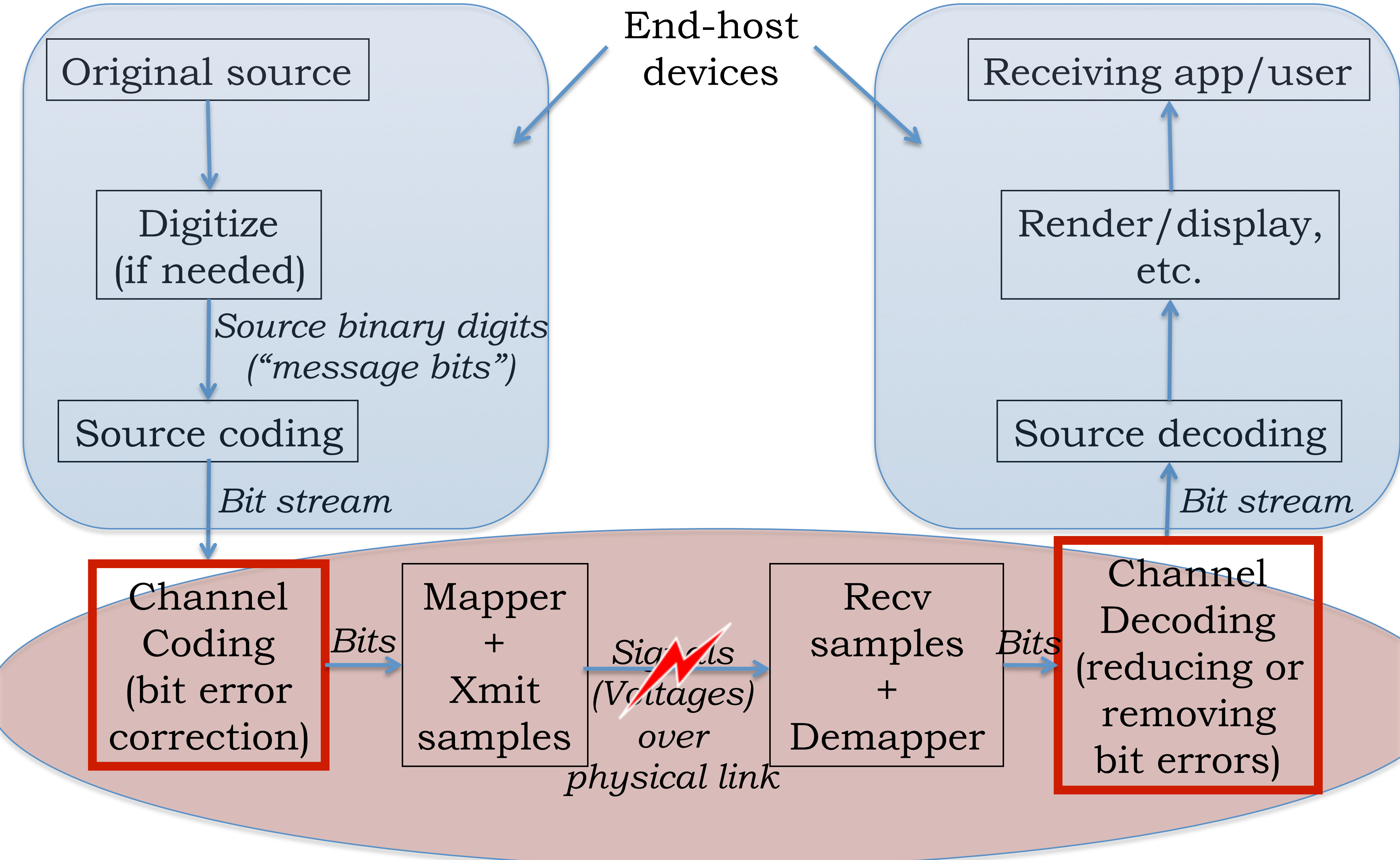
INTRODUCTION TO EECS II

DIGITAL COMMUNICATION SYSTEMS

6.02 Fall 2013
Lecture #4

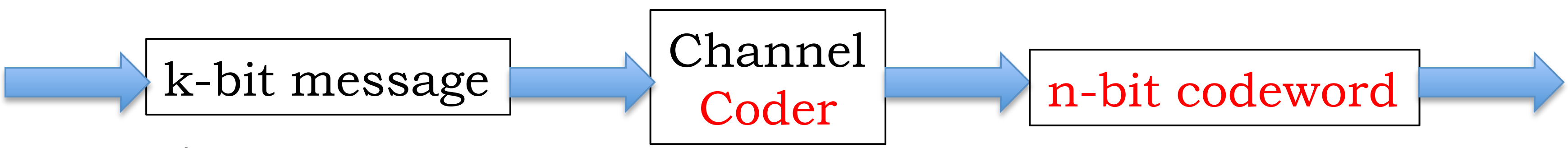
- Linear block codes for channel coding
  - Rectangular codes
  - Hamming codes

Single Link Communication Model



Channel Coding

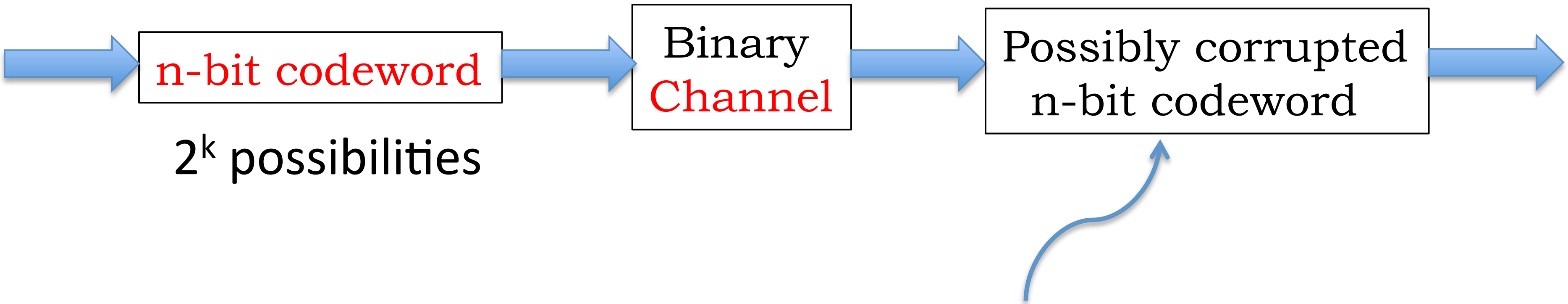
Block code: Block of ***k*** message bits at a time is encoded to **n > k** code bits, with each of the $\mathbf{2^{k}}$ possible messages encoded into a unique **n-bit** codeword



2^k possibilities

2^k codewords out of 2^n possibilities

Channel Transmission

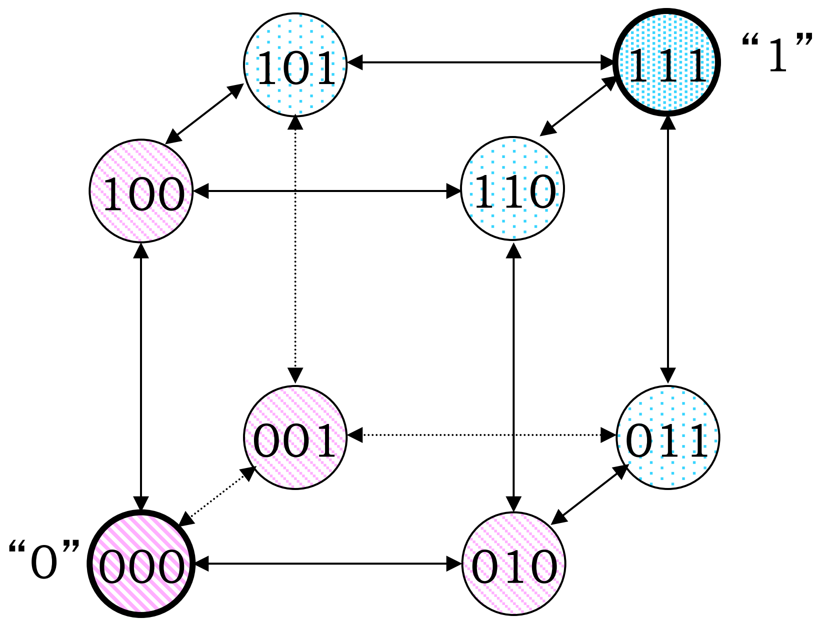


Only 2^k out of 2^n possibilities are valid, others are corrected to nearest* (in HD) valid neighbor

(*provided channel's probability of a bit flip is $p < 0.5$)

Embedding for Structural Separation

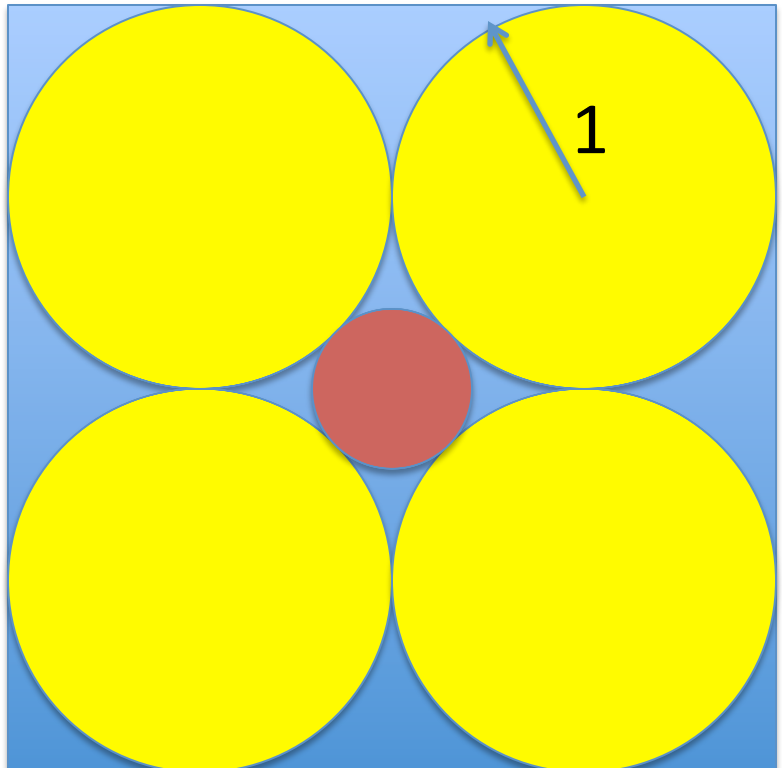
- Encode so that the codewords are far enough from each other – likely error patterns shouldn't transform one codeword to another.
- How much volume to allow around each codeword depends on the likely level of noise.



Code: A choice of 2^k out of 2^n nodes; a one-to-one mapping of all k-bit message strings to n-bit *codewords.*

The ***code rate*** is **k/n**. If **min HD** = **d**, then this is an **(n,k,d)** code.

Our intuition for n ≥ 4 dimensions



Extending this construction to n dimensions,
we get a red hypersphere confined within a blue
hypercube by surrounding touching yellow
unit-radius hyperspheres. Yes? No?

No!!

For n ≥ 10, the red sphere is no longer confined to the hypercube, because then its radius

$$\sqrt{n} - 1 > 2$$

In fact, the fraction of its volume inside the cube goes down exponentially fast with increasing n.

From *The Cauchy-Schwarz Master Class* by J. Michael Steele
http://www-stat.wharton.upenn.edu/~steele/Publications/Books/CSMC/CSMC_index.html

Minimum Hamming Distance of Code vs. Detection & Correction Capabilities

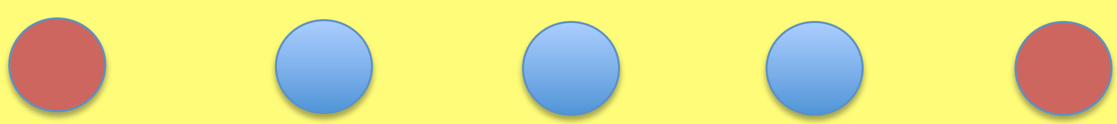
If d is the minimum Hamming distance between codewords, we can:

- 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.:



$d=4$,
$t_C=1$, $t_D=2$

A Simple Code for Single-Error Detection: Parity Check

- Add a parity bit P to message of k data bits $\{D_i\}$ to make the total number of "1" bits even (aka "even parity"). Can compute P as

  $P = D_1 + D_2 + \ldots + D_k$ => addition in GF(2), i.e., binary/Boolean arithmetic

- If the number of "1"s in the received word is *odd*, there there has been an error:

  0 1 1 0 0 1 0 1 0 0 1 1 → original word with parity bit
  0 1 1 0 0 0 0 1 0 0 1 1 → single-bit error (detected)
  0 1 1 0 0 0 1 1 0 0 1 1 → 2-bit error (not detected)

- Minimum Hamming distance of parity check code is 2 (proof?)
  - Detect all single-bit errors

    (detect any odd number of errors, no even number of errors)
  - Cannot correct any errors

Without additional structure ···

- Hard to
  - Design a good code (for large minimum HD between codewords, or other criteria)
  - Decode (each received n-bit word requires 2^k comparisons of the received n-bit word with those in the dictionary of valid codewords)

How to Construct Codes?

0000000	1100001	1100110	0000111
0101010	1001011	1001100	0101101
1010010	0110011	0110100	1010101
1111000	0011001	0011110	1111111

Want: 4-bit messages with single-error correction (min HD=3)

How to produce a code, i.e., a set of codewords, with this property?

Linear Block Codes

Linear block code: … codewords obtained via a *linear transformation* of the message bits.

Key property: Sum of any two codewords is *also* a codeword.

This is necessary and sufficient for a code to be linear. Hence:

- **All – "0" codeword** is always in a linear code.
- **Min HD**: Smallest weight (i.e., number of "1"s) among nonzero codewords.

Generator Matrix of Linear Block Code

Linear transformation:

$$\mathbf{c} = \mathbf{d}.\mathbf{G}$$

c: codeword (n-element row vector)

d: data/message (k-element row vector)

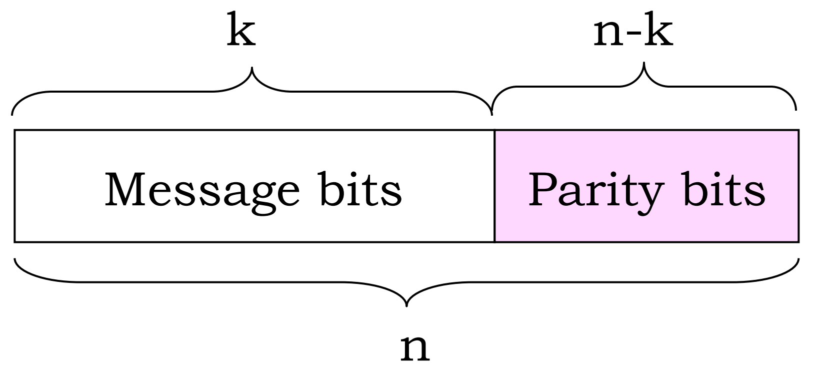
G: *generator matrix* (k rows, n columns)

c is a linear combination of rows of **G**, weighted by the corresponding message bits in **d**

$\mathbf{c}_j$ is a linear combination of the message bits in **d**, weighted by the corresponding entries in the j-th column of **G**

(n,k) Systematic Linear Block Codes

- *k*-bit blocks
- Add (*n*-*k*) generalized parity bits to each block



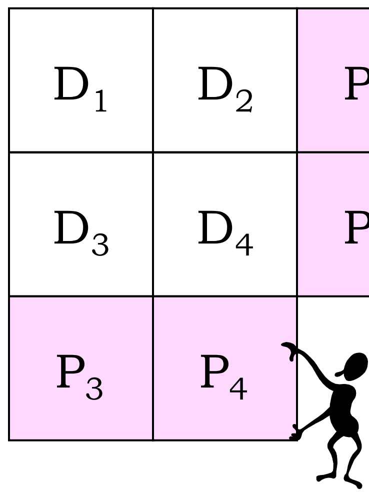
Every linear code can be represented by an equivalent systematic form --- ordering is not significant, direct inclusion of k message bits in n-bit codeword is.

Corresponds to using invertible transformations on rows, and permutations on columns, to get

G = [**I** | **A**] --- identity matrix in the first k columns

What is **A** for the simple parity check code?

Example of Generalized Parity Checks: Rectangular Parity Codes



P_1 is parity bit for row #1

$(n,k,d)=?$

P_4 is parity bit for column #2

0 1 1
1 1 0
1 0

Parity for each row and column is correct ⇒ no errors

0 1 1
1 0 0
1 0

Parity check fails for row #2 and column #2 ⇒ bit D_4 is incorrect

0 1 1
1 1 1
1 0

Parity check only fails for row #2 ⇒ bit P_2 is incorrect

Anything else:
⇒ "uncorrectable error"

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.

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

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

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

Generator Matrix for (9,4,4) Rectangular Code

For the (9,4,4) rectangular code that includes an overall parity bit:

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

1×k message vector

k×n generator matrix

1×n code word vector

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

Some practice

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 We Really Need?

- **n-k** parity bits can represent $\mathbf{2^{n-k}}$ possibilities

- For **single-bit error** correction, parity bits need to represent n+1 possibilities:
  - No error
  - Error in i-th bit out of n-bit codeword

- So $\mathbf{n+1 \leq 2^{n-k}}$ or

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

- 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)
- $(2^m - 1, 2^m - 1 - m, 3)$

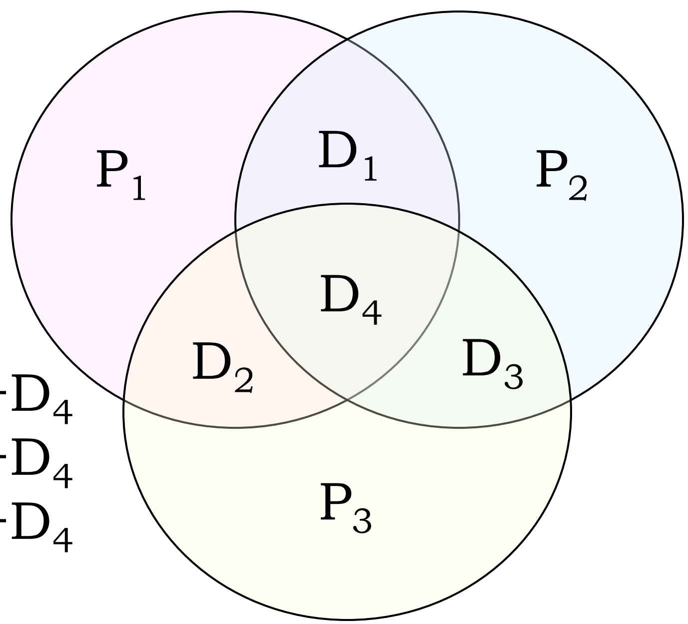
- Such efficiency is not the only, or even most important, criterion in picking a good code. The ability of a code's k/n to approach channel capacity, and various other factors, are important.

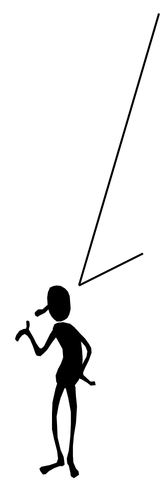
(7,4,3) Hamming Code Example

- 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

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





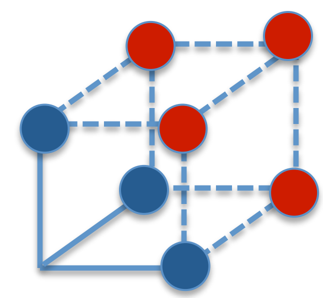
Suppose we check the parity and discover that P1 and P3 indicate an error?
bit D2 must have flipped

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

$E_3E_2E_1$	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

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

Constraints for more than single-bit errors

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

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

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) … (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