

Compression

- **Lossless**

- Data received = data sent
- Used for executables, text files, numeric data

- **Lossy**

- Data received \approx data sent
- Used for images, video, audio
- Takes advantage of human perception

Simple lossless algorithms

- **Run Length Encoding (RLE)**

- Example: AAABBCDDDD → 3A2B1C4D
- Good for scanned text (faxes, 8-to-1 compression ratio)
- Can increase size for data without much repetition
- Example: AABCCDE → 2A1B2C1D1E

- **Differential Pulse Code Modulation (DPCM)**

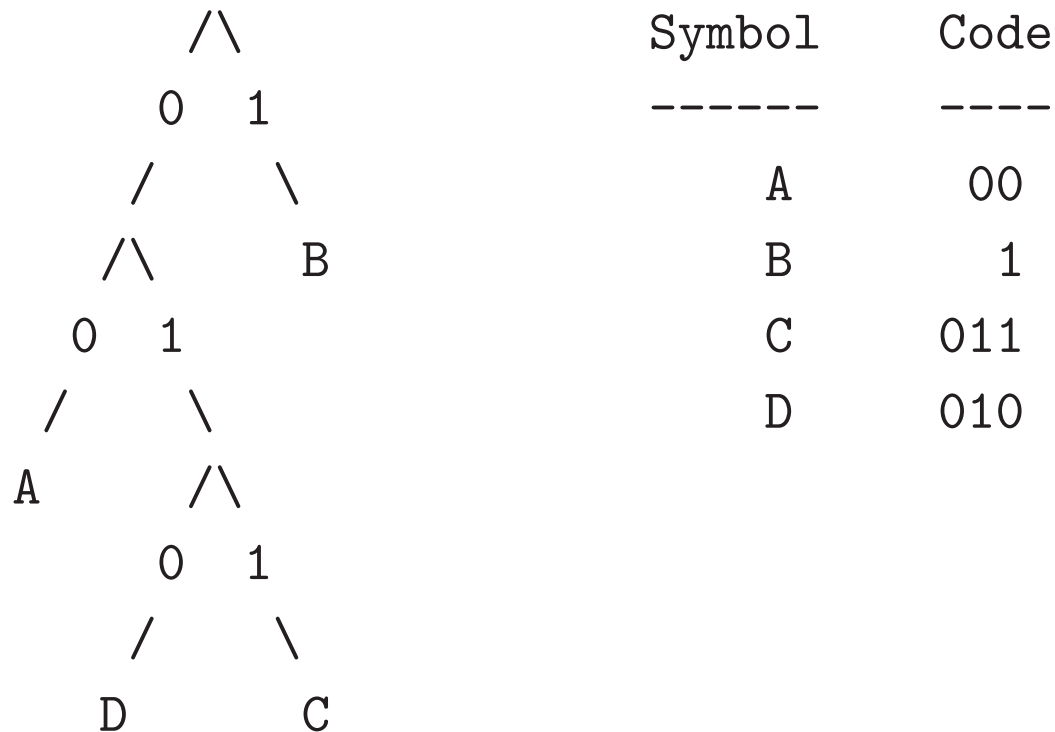
- Example: AAABBCDDDD → A0001123333
- Deltas encoded with fewer bits than symbols.
- Change reference symbol if delta becomes too large.
- Works better than RLE for many digital images (1.5-to-1).

Huffman coding

- **Consider data as sequence of symbols**
- **Suppose you know the frequency of each symbol**
 - Assign symbols *weight* proportional to frequency
- **Use shorter code for more frequent characters**
 - Consider a phone at Brown. Each digit is a symbol
 - Most numbers dialed within Brown are 1-401-86x-xxxx
 - So compress phone number, by just dialing x-xxxx
 - 1-401-yyy-yyyy \rightarrow 8-yyy-yyyy
 - But, 1-617-zzz-zzzz \rightarrow 8-1-617-zzz-zzzz
(longer code, but less frequently used)
- **Usually, have smaller symbol alphabet**
 - E.g., symbols are bytes, codes are in bits

Huffman trees

- Can view Huffman codes as paths in a tree
 - Symbols with heavier weight are higher in tree
- Prefix-free: No code is another's prefix.
- Example: $B = 16$, $A = 8$, $C = 4$, $D = 4$



Building Huffman trees

- **Given weights, can build a suitable Huffman tree**
 - Take two symbols with least weight
 - Assign suffix "0" to one, suffix "1" to the other
 - Combined both symbols and weights and continue
- **Example:** $B = 16$, $A = 8$, $C = 4$, $D = 4$
 - Let suffix of D be "0", of C be "1".
 - New weights: $B = 16$, $A = 8$, $CD = 8$
 - Next two are A and $CD = 8$; give A suffix "0"
 - Now give ACD suffix "0", B suffix "1"
 - Result is tree from the previous slide.

Transmitting Huffman trees

- **Decoder needs Huffman tree**
- **One approach: Hard-code tree in encoder/decoder**
 - *E.g.*, could come up with suitable weights for English text
- **Another approach: Include tree with message**
 - Deterministically agree on one tree per set of weights:
 - All codes of a given bit length have lexicographically consecutive values, in the same order as the symbols they represent;
 - Shorter codes lexicographically precede longer codes
 - **Now just need to transmit length of each symbol's code**
(which can be further compressed)

Arithmetic coding

- **Like Huffman, assume symbol weights known**
- **Assign each symbol a range in $[0, 1)$**
 - Example: Say weights are $B = 16$, $A = 8$, $C = 4$, $D = 4$
 $B = [0, 0.5)$, $A = [0.5, 0.75)$, $C = [0.75, 0.875)$, $D = [0.875, 1)$
 - Now for each symbol, subdivide the range
 - For String "BBA", ranges will subdivide to get:
 $[0, 0.5)$, $[0, 0.25)$, $[0.125, 0.1875)$
 - Message is transmitted as a number in the final range.
- **Potentially better than Huffman codes**
 - Can better accommodate non-power-of-two weights
 - Can better model data dependent weights.
 - Less often used (patents, complexity/time/space)

Dictionary-Based Methods

- **Build dictionary of common terms**
 - Variable length strings
- **Transmit index into dictionary for each term**
- **Lempel-Ziv (LZ) is the best-known example**
- **Commonly achieve 2-to-1 ratio on text**
- **Variation of LZ used to compress GIF images**
 - First reduce 24-bit color to 8-bit color (lossy if more than 256 colors in original)
 - Treat common sequence of pixels as terms in dictionary
 - Not uncommon to achieve 10-to-1 compression (x3)

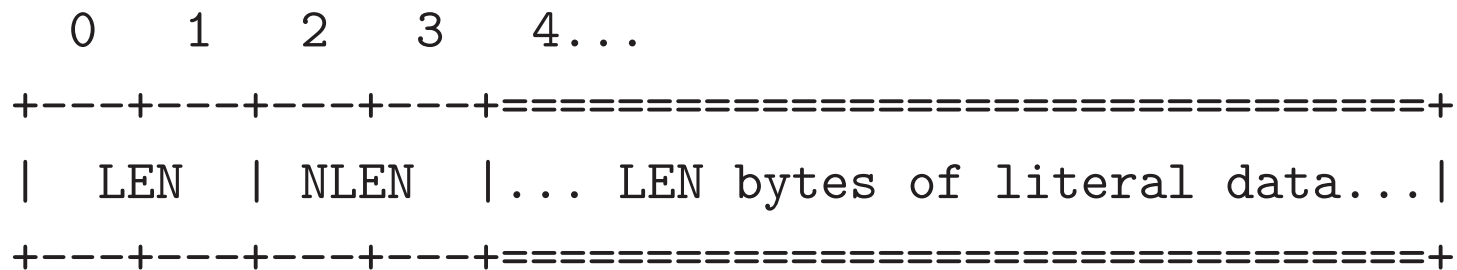
Gzip compression (LZ77)

- **Big idea: Include backreferences to past data**
- Consider string “Blah blah blah blah blah!”^a
- Notice repeat: “Blah blah blah blah!”
- Encode “Blah blah b” as “Blah b[$L = 5, D = 5$]”
 - Copy string of L bytes, from D characters back.
- **But why stop there. Can say:**
“Blah b[$L = 18, D = 5$]!”
 - Length L can extend into copied string!

^aexample from gzip site

Gzip details - gzip.org/deflate.html

- Compressed file is composed of blocks of data
- First bit is **BFINAL** – set if final block
- Next two bits are **BTYPE**
 - 00 – no compression (if compression makes things worse)
 - 01 – compressed with fixed Huffman codes
 - 10 – compressed with dynamic Huffman codes
 - 11 – reserved (error)
- **No compression blocks (up to $2^{16} - 1$ Bytes):**



(NLEN is one's compliment of LEN)

Huffman coding of gzip

- **Use two Huffman trees**
 - One tree for symbols & length (L) values
 - Second tree for back distances (D)
 - Note L value always followed by D , so no ambiguity
- **Symbol/length alphabet has 285 symbols**
 - 0–255 → are data bytes
 - 256 → end of block
 - 257–285 → length of backreference
- $L \in \{3, \dots, 258\}$ **How to encode in 29 values?**

Encoding L

- Add extra bits after some symbols

- E.g., Symbol 264 means $L = 10$
- 265 means $L \in \{11, 12\}$, extra bit follows

	Extra			Extra			Extra	
Code	Bits	Length(s)	Code	Bits	Lengths	Code	Bits	Length(s)
-----	-----	-----	-----	-----	-----	-----	-----	-----
257	0	3	267	1	15,16	277	4	67-82
258	0	4	268	1	17,18	278	4	83-98
259	0	5	269	2	19-22	279	4	99-114
260	0	6	270	2	23-26	280	4	115-130
261	0	7	271	2	27-30	281	5	131-162
262	0	8	272	2	31-34	282	5	163-194
263	0	9	273	3	35-42	283	5	195-226
264	0	10	274	3	43-50	284	5	227-257
265	1	11,12	275	3	51-58	285	0	258
266	1	13,14	276	3	59-66			

Encoding D

- Similar scheme for D , which has 30 symbols

	Extra			Extra			Extra	
Code	Bits	Dist	Code	Bits	Dist	Code	Bits	Distance
-----	-----	-----	-----	-----	-----	-----	-----	-----
0	0	1	10	4	33-48	20	9	1025-1536
1	0	2	11	4	49-64	21	9	1537-2048
2	0	3	12	5	65-96	22	10	2049-3072
3	0	4	13	5	97-128	23	10	3073-4096
4	1	5,6	14	6	129-192	24	11	4097-6144
5	1	7,8	15	6	193-256	25	11	6145-8192
6	2	9-12	16	7	257-384	26	12	8193-12288
7	2	13-16	17	7	385-512	27	12	12289-16384
8	3	17-24	18	8	513-768	28	13	16385-24576
9	3	25-32	19	8	769-1024	29	13	24577-32768

Gzip's fixed Huffman codes

- *D* values just encoded as 5 bits
 - Values 30 and 31 never used
- Symbol/*L* values use following code:

Lit Value	Bits	Codes
-----	-----	-----
0 - 143	8	00110000 - 10111111
144 - 255	9	110010000 - 111111111
256 - 279	7	0000000 - 0010111
280 - 287	8	11000000 - 11000111

- Why might these be a good choice?

Dynamic Huffman codes

- **Must first transmit two Huffman trees (symb/ L & D)**
 - Recall means transmitting code length for each symbol
 - Use code length 0 for unused symbols
 - Include length of table, truncate after last non-0
- **Use Huffman + RLE on the Huffman trees! Alphabet:**

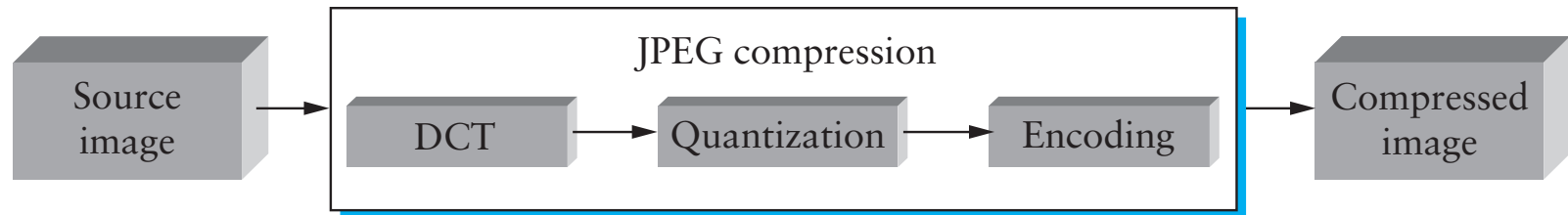
	Code	Extra Bits	Meaning
	-----	-----	-----
0 - 15	0	0	Represent code lengths of 0 - 15
16	2	2	Copy the previous code length 3 - 6 times
17	3	3	Copy previous code length 3 - 10 times
18	7	7	Copy previous code length 11 - 138 times

- Must transmit 3-bit code length for each of 19 symbols (0–18)
- Transmit lengths in order that often helps truncation:
16, 17, 18, 0, 8, 7, 9, 6, 10, 5, 11, 4, 12, 3, 13, 2, 14, 1, 15

Differential Dictionary Methods

- **Idea: Don't send data that is at the receiver already.**
 - Copying a new version of a file over an old.
- **Simplistic: Checksums, timestamps on files.**
- **Better: Checksums on blocks**
- **Better yet: Data dependent blocks**
 - Rolling Checksums
 - Calculate checksum on 0-512, then 1-513, etc.
 - rsync: When checksum = 0 (mod k), delimit block
 - Rabin Fingerprints are fast *and* collision resistant.

Lossy image compression



- **JPEG: Joint Photographic Expert Group (ISO/ITU)**
- **Lossy still-image compression**
- **Three phase process**
 - Process in 8x8 block chunks (macro-block)
 - DCT: transforms signal from spatial domain into and equivalent signal in the frequency domain (loss-less)
 - Apply a quantization to the results (lossy)
 - RLE (for 0s) + Huffman encoding (loss-less)
 - For color: each pixel is three values (YUV)
($Y = \text{luminescence}$, $U = c_1(B - Y)$, $V = c_2(R - Y)$)

- **Original Data**

$$\begin{pmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{pmatrix}$$

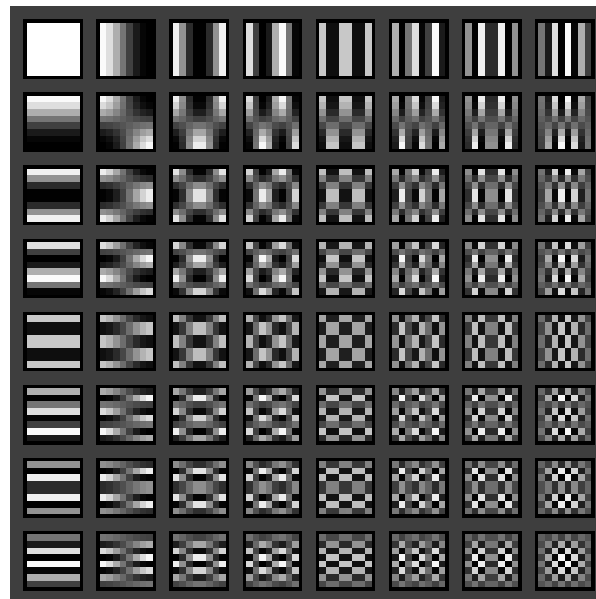
- **Subtract 128**

$$\begin{pmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{pmatrix}$$

- **Discrete Cosine Transformation (rounded)**

$$\begin{pmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{pmatrix}$$

- **What do DCT entries actually *mean*?**



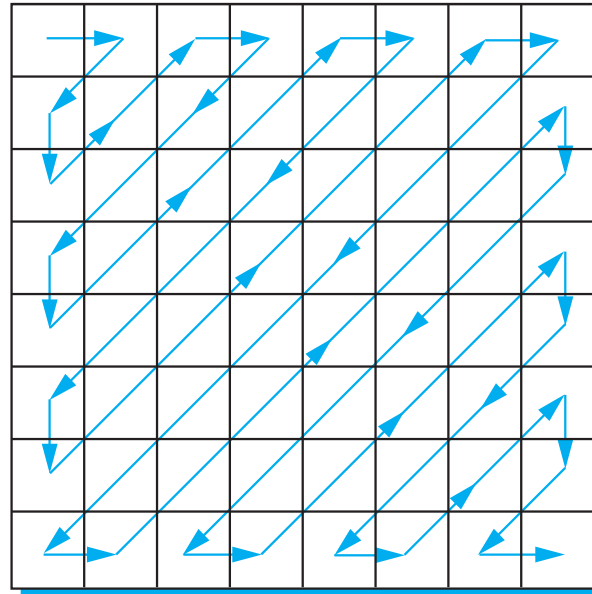
- **Quantization Table (the lossy part)**

$$\begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

- **Quantized DCT (to be encoded losslessly)**

$$\begin{pmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- **Encoding Pattern**



- **Encode: -26, -3, 0, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1, EOB**
- **Huffman, or Arithmetic codes.**

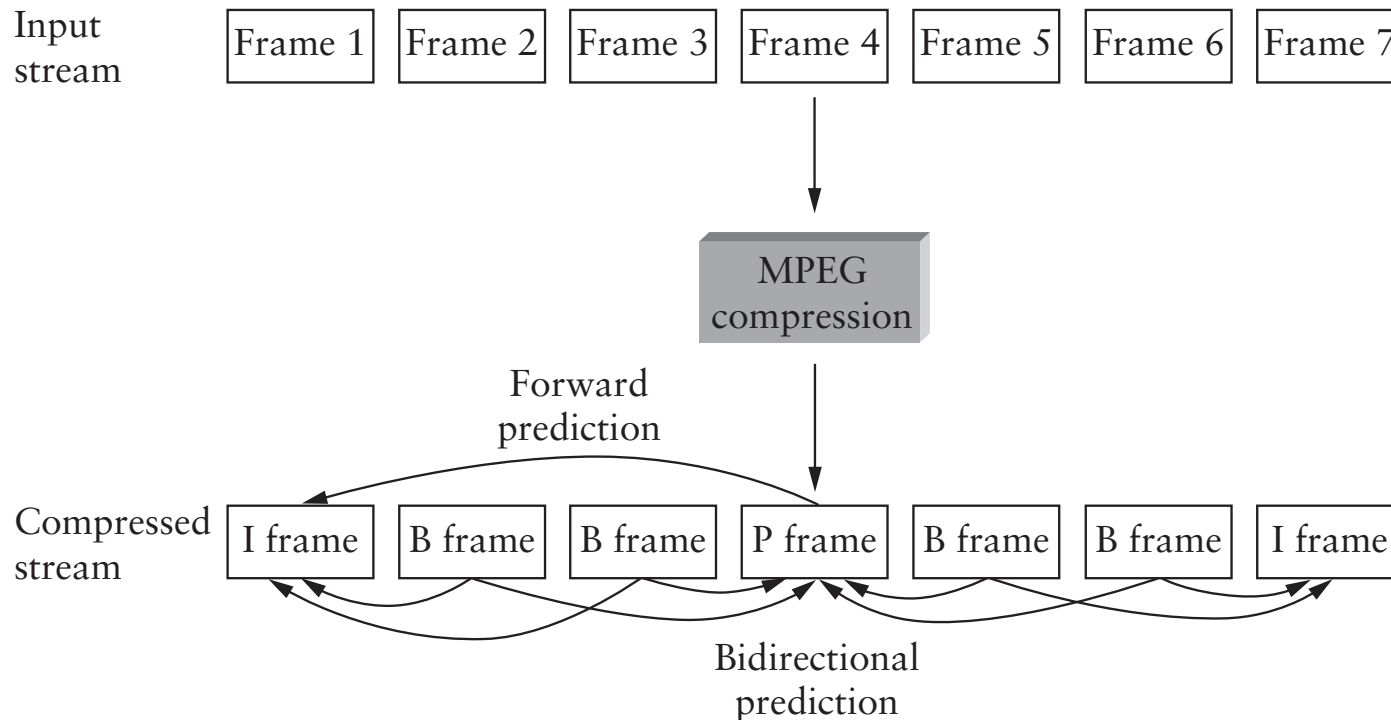
MPEG

- **Motion Picture Expert Group**
- **Lossy compression of video**
- **First approximation: JPEG on each frame**
 - Y value encoded in 16×16 “macroblock”
 - U, V channels downsampled into 8×8 macroblocks
- **Also remove inter-frame redundancy**
 - *E.g.*, when panning, or objects move across image

MPEG (cont)

- **Frame types**

- I frames: intrapicture
- P frames: predicted picture
- B frames: bidirectional predicted picture



- **Example sequence transmitted as I P B B I B B**

MPEG (cont)

- **B and P frames**

- Coordinate for the macroblock in the frame
- Motion vector relative to previous reference frame (B, P)
- Motion vector relative to subsequent reference frame (B)
- Delta for each pixel in the macro block

- **Effectiveness**

- Typically 90-to-1
- As high as 150-to-1
- 30-to-1 for I frames
- P and B frames get another 3 to 5×

Transmitting MPEG

- **Adapt the encoding**
 - Resolution
 - Frame rate
 - Quantization table
 - Specified in GOP (group of pictures) header
- **Packetization (best to lose whole frames)**
- **Dealing with loss**
- **GOP-induced latency**

MP3

- **CD Quality**

- 44.1 kHz sampling rate
- $2 \times 44.1 \times 1000 \times 16 = 1.41$ Mbps
- Actually requires more for synchronization & error correction

- **Strategy**

- Split into some number of frequency bands
- Divide each subband into a sequence of blocks
- Encode each block using DCT + Quantization + Huffman
- Trick: how many bits assigned to each subband

Coming up

- **This week: TCP connection with another group**
- **April 9: HW3 out**
- **April 16: HW3 due**
- **April 20: TCP Due**