The Magic of Data Compression

"From something to nothing and back again"

David Gow

2013

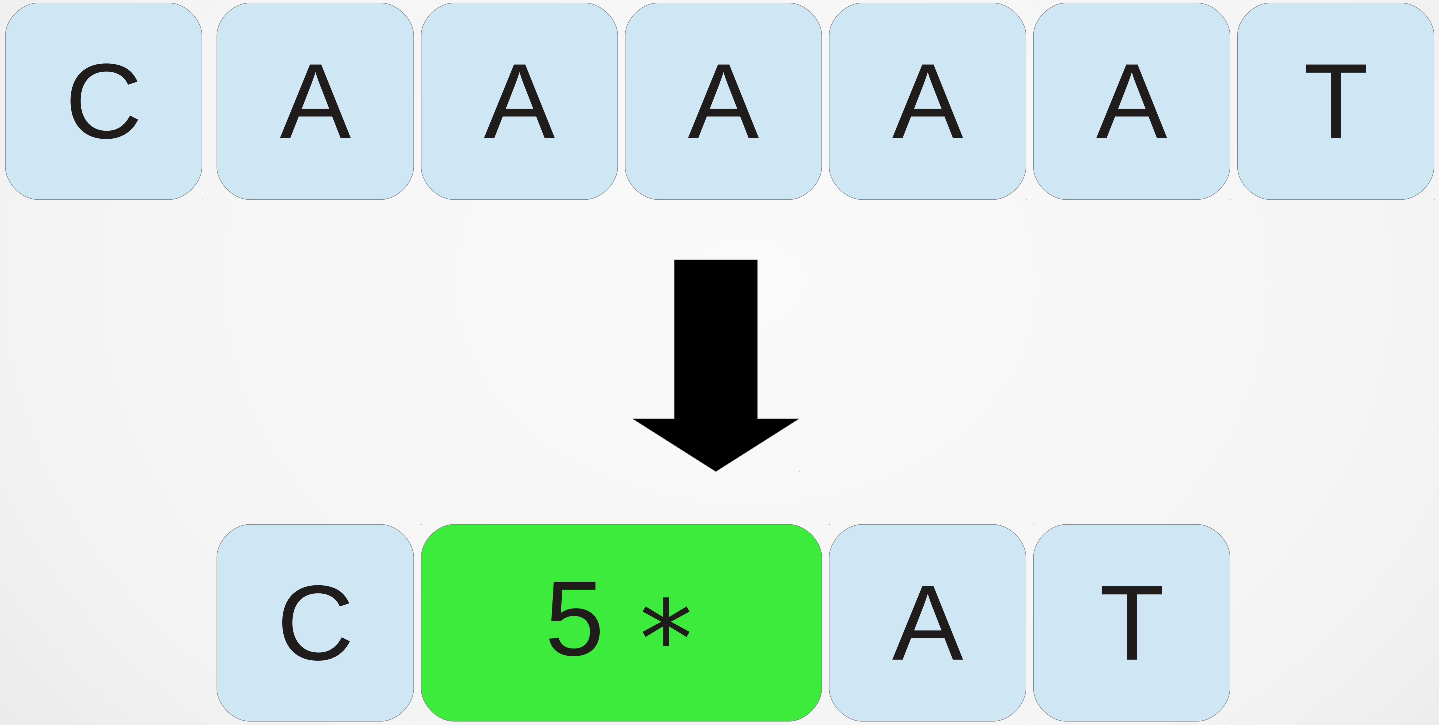
Lossless Compression

- The input is exactly the same as the output: no information is lost.
- “Information Theory”
- Find and elminiate redundancy
- NOT ALL DATA IS COMPRESSABLE!

RLE

- Run Length Encoding
- Very simple
- Good at compressing “runs” of repeated characters
- Can have different unit sizes: 8-bit, 16-bit

RLE



RLE Decoding

- → next
- if (next == marker)
  - → count
  - if count == 0
    - ←marker
  - else
    - → value
    - ← value (count times)
- else
  - ← next

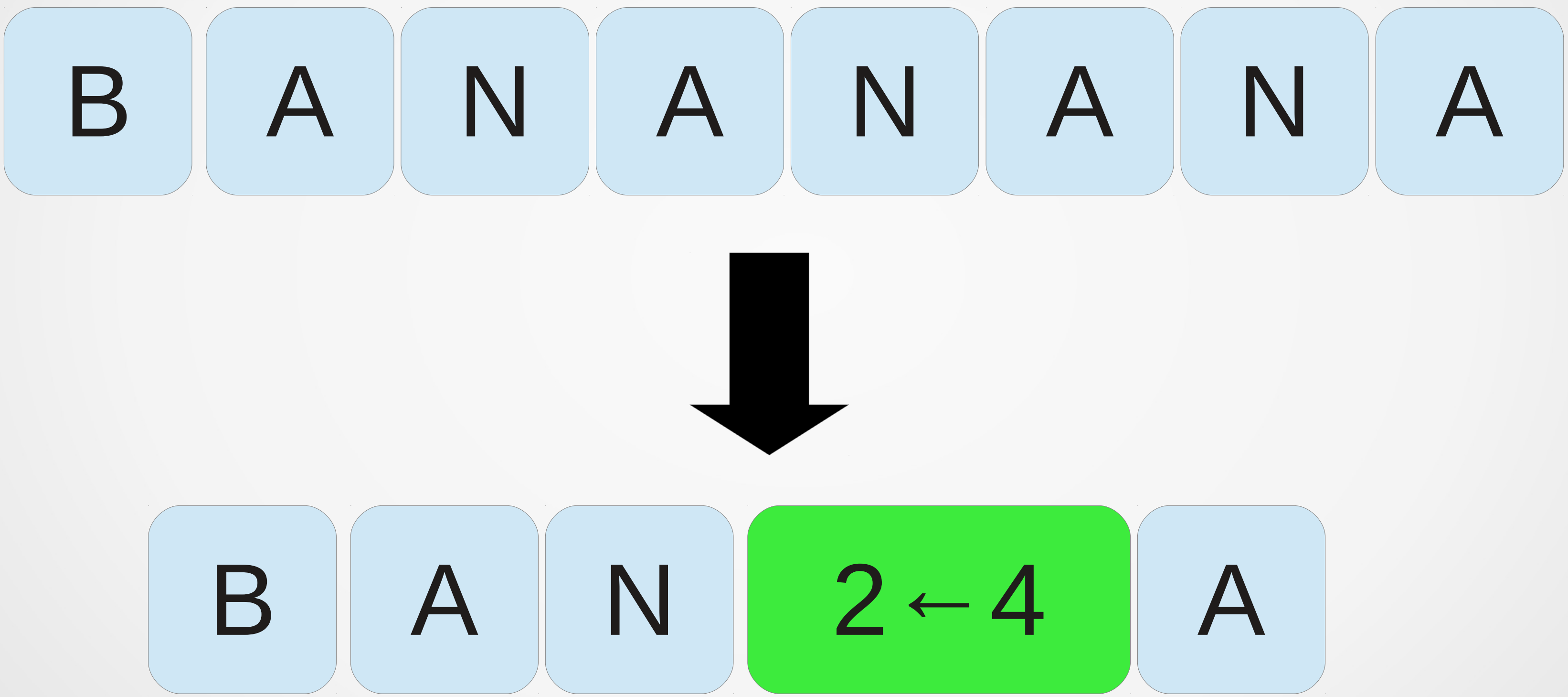
RLE Encoding

- →value
- if (value == prev)
  - count++
- else
  - if (count > 3)
    - ← marker
    - ← count
    - ← value
  - else
    - ← value (count times)
  - prev = value
  - count = 0

LZ

- Lempel-Ziv
- Use words from a dictionary or back-references
- Lots of different varieties
  - LZ77 (This is the one we'll be looking at)
  - LZ78 (Dictionary)
  - LZW (Complicated LZ78, used in .gif)
  - LZMA (7zip, markov chains)

LZ



LZ Decoding

- →value
- if (value == marker)
  - → [offset,count]
  - if (count = 0)
    - output marker
  - else
    - copy count bytes at offset
- else
  - output marker

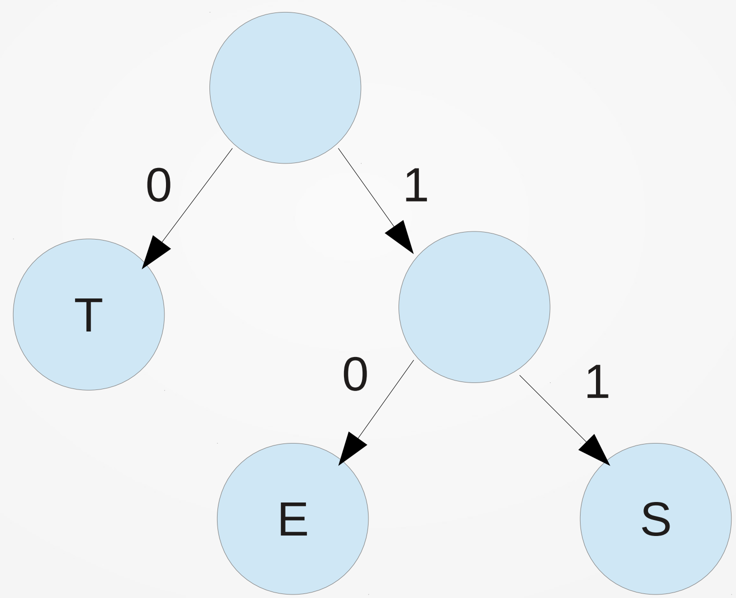
LZ Encoding

- Concept of a sliding-window
- Look back N bytes and search for the best sequence to copy
- There are some clever ways of speeding this up
- Very simple LZ implementation (in C): http://www.ucc.asn.au/tech/2013/0x01_sulix/fastlz.c

Huffman

- Entropy coding
- Choose code size based on frequency
- e.g. E → 3 bits, Z → 11 bits
- Has a 'dictionary'
  - Also known as a table or tree
  - It's generated as a tree
- See also: Arithmetic coding, Range coding

Huffman



TEST = 0 10 11 0

Huffman: Building trees

- Count all of the bytes in the source (or reference) data
  - You need the frequencies that characters occur
- Take the two nodes with the lowest frequencies and 'merge' them
  - Replace them in the list with a single node that has both original nodes as children
- Repeat: you'll end up with the optimal huffman tree.

Huffman: Decoding

- → bit
- if bit == 0:
  - currentNode = currentNode.left
- else:
  - currentNode = currentNode.right
- if currentNode.character:
  - ← currentNode.characyer
  - currentNode = rootNode

Huffman Encoding

- Build a <character → bits> map
  - Traverse the tree backwards.
- Loop through characters in input and output corresponding bits
- Don't forget to make sure the encoder and decoder have the same dictionary.
- See the code here: http://www.ucc.asn.au/tech/2013/0x01_sulix/huff.c
  - The code that got me into compression! :)

Deflate

- Not going into this in detail
- Basically LZ + Huffman
- A few different 'block' methods:
  - Uncompressed
  - LZ + Huffman with pre-arragned dictionary
  - LZ + Huffman with embedded dictionary
- Used in zip, zlib, gzip, png, “the web” and pretty much everything you've ever heard of.

Lossy Compression

- Loses some information about the input file
- Try to remove bits which people don't notice
- Used in media
  - MP3/AAC/Ogg Vorbis
  - JPEG
  - MPEG/h.264
  - and friends!

MP2 (Roughly)

- Predecessor to MP3
- Take an audio stream and split it into “frames” a few tenths of a second long
- Split each frame into 32 frequency bands.
- Remove the frequency bands that are difficult to hear.
  - The ones with the lowest “power”
  - The ones which are too high for the human ear
  - The ones which are “masked” by nearby powerful bands
- A “psychoacoustic” model

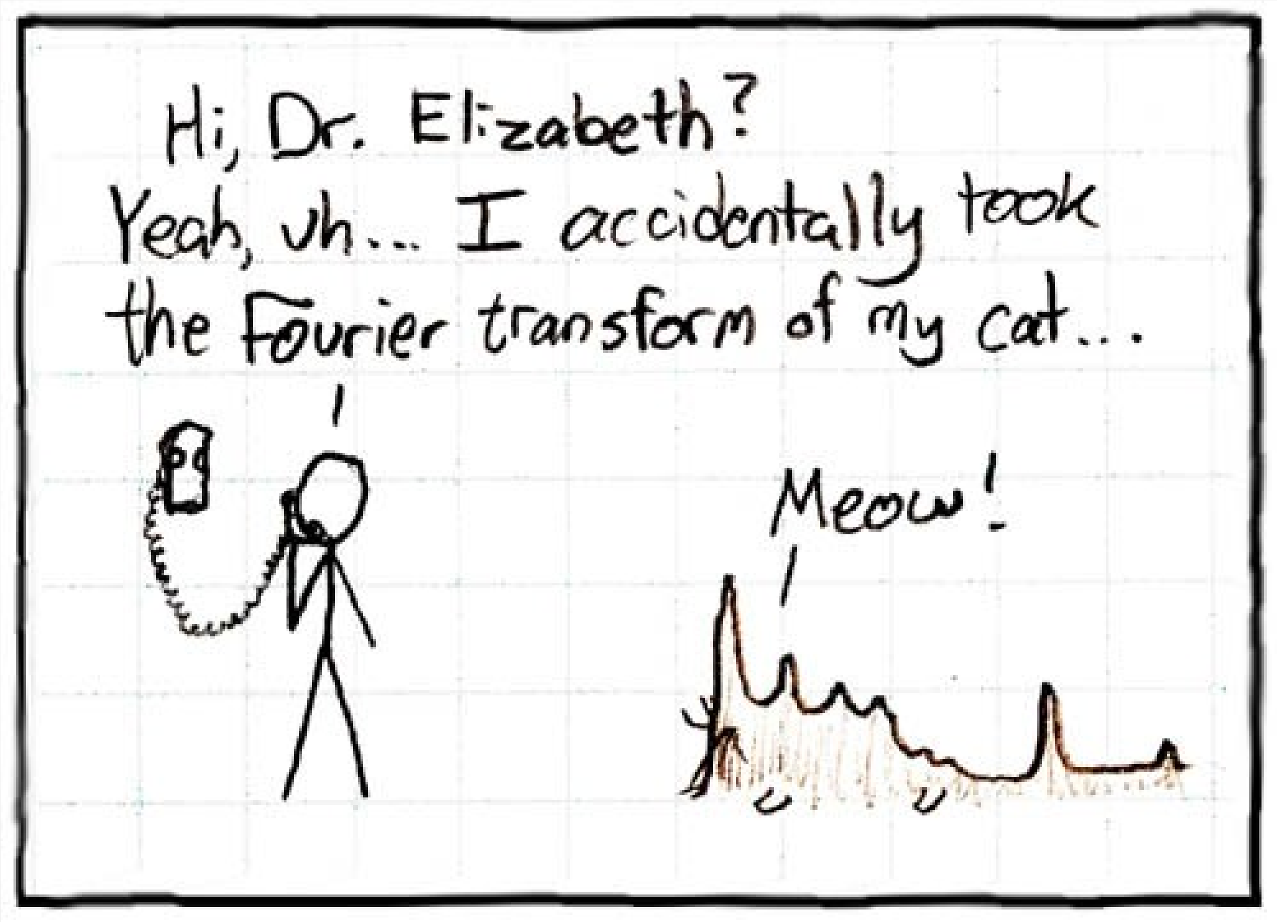
Fourier: The Frequency Domain

- Frequency is a better match for the human ears than time (to a point)
- Convert 'frames' entirely to be a function of frequency
- The Fourier Transform

$$\int_{-\infty}^{\infty} f(x) e^{-2\pi i x \zeta} dx$$

- Basically finding coefficients for sin() and cosine() functions

Fourier: The Frequency Domain



That cat has some serious periodic components

Fourier vs Cosine!

- The Fourier transform is good!
- But it doesn't “constrain power” to the lower frequencies well.
- This makes it less efficient for codecs like JPEG and Ogg Vorbis
- Use the Discrete Cosine Transform (DCT) instead

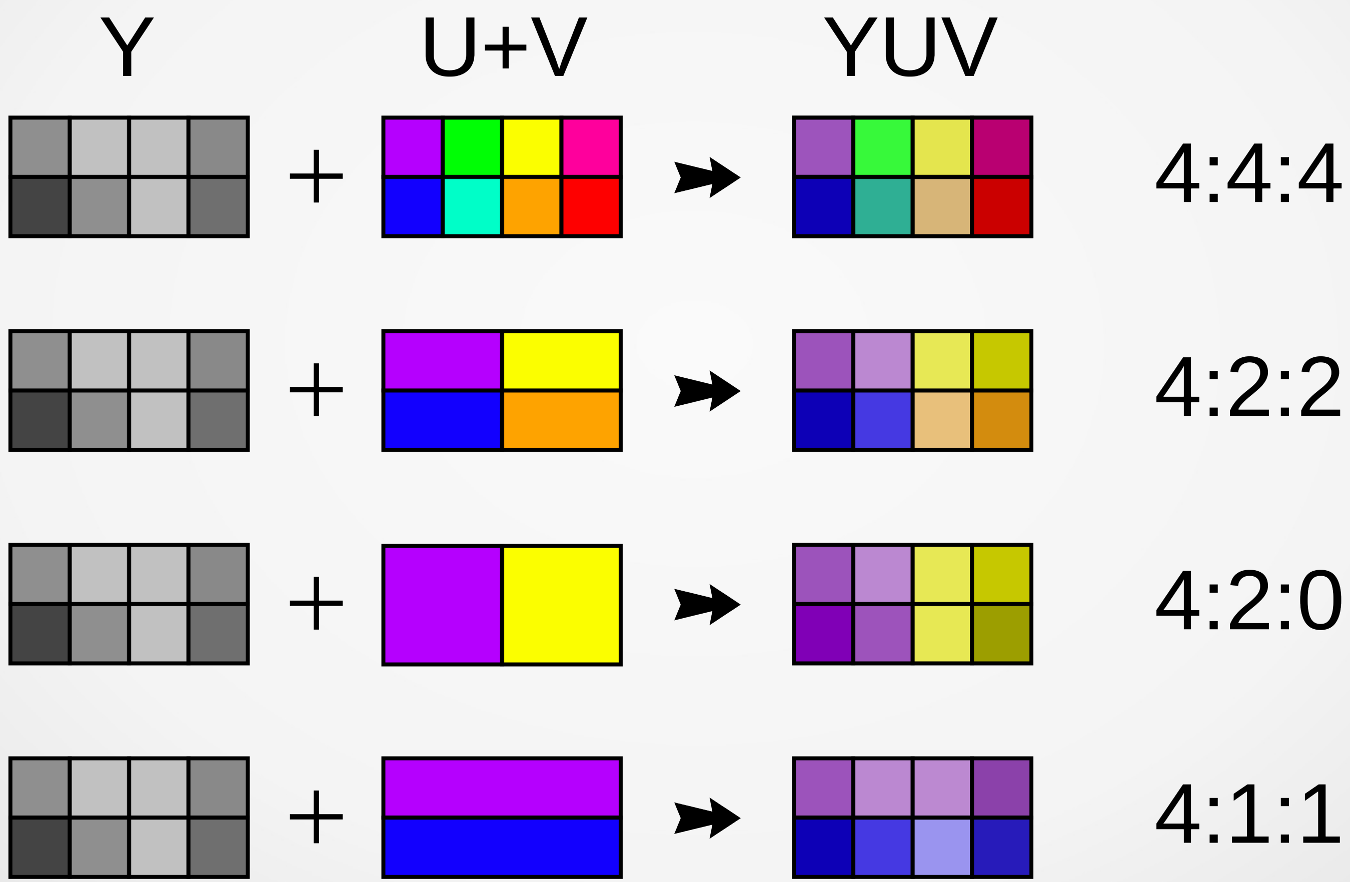
JPEG

- Switching tracks from audio to image
- JPEG: named after its creators:
  - Joint Photographic Experts Group
  - Designed for photographs

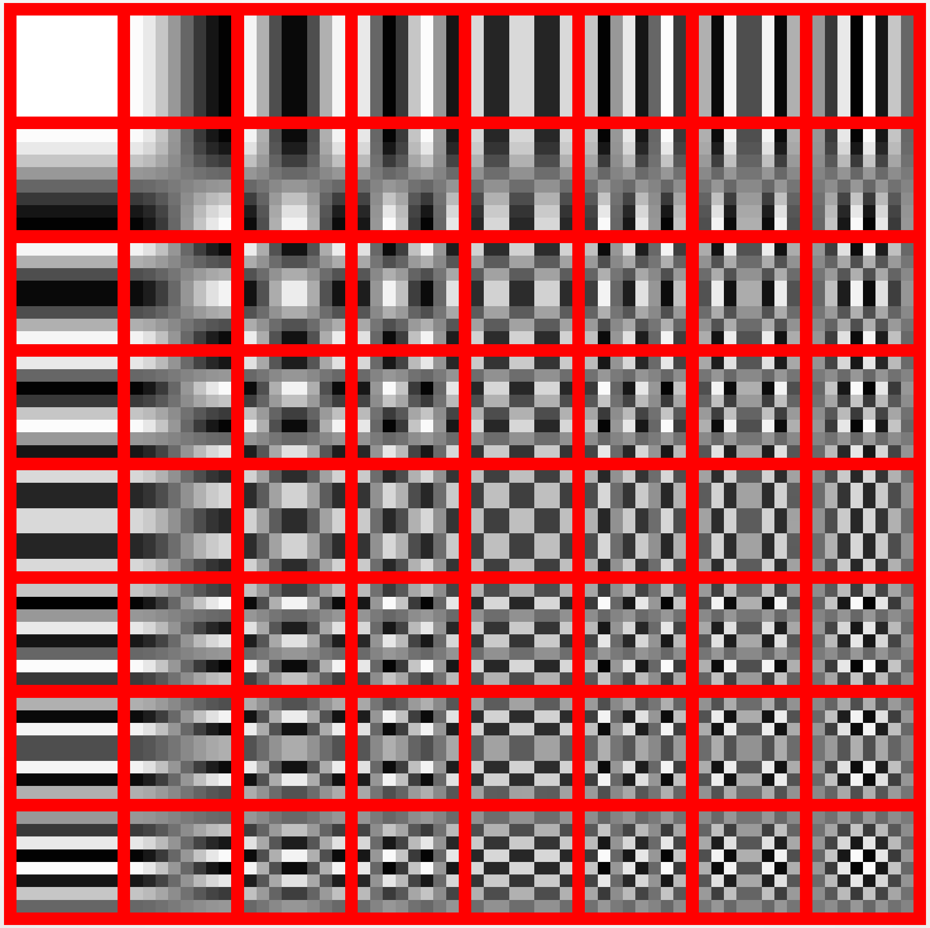
JPEG

- Break the image up into 16x16 px squares (macroblocks)
- Break up each macroblock into SIX 8x8 pixel squares (blocks)
  - 4 greyscale
  - 2 colour (scaled)
- Each block undergoes the 2D DCT
- Frequency values are quantized
  - This is the actual lossy compression bit
- Final bitstream is Huffman compressed

JPEG: Chroma Subsampling



JPEG: DCT



JPEG

- Quantization is greater for higher frequencies
  - The human eye picks up on them less than lower frequencies
- One can sometimes see “blocking” artefacts when a JPEG is stored in low quality.
- Also “ringing” artefacts when too much power is given to high-frequency components.

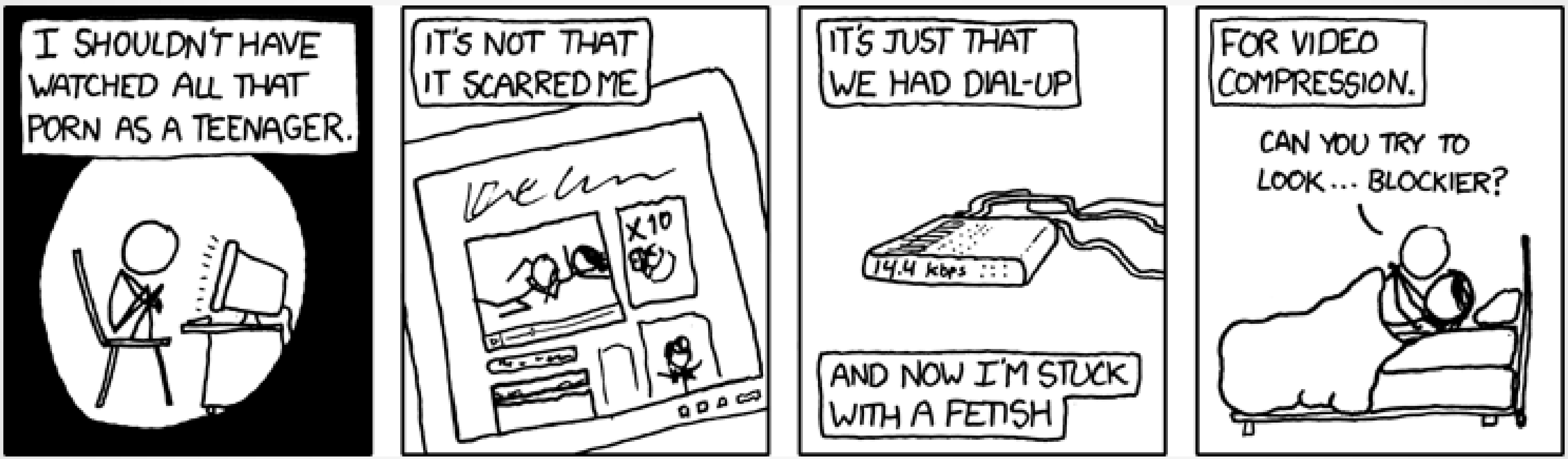
Ogg Vorbis

- Ogg Vorbis
  - Open source audio codec (coder-decoder)
  - Vorbis is the actual audio codec, Ogg is a “container”
- Uses a MDCT (Modified Discrete Cosine Transform) on OVERLAPPING audio frames
- Frames are quantized.
- Similar psy optimizations to MP2/MP3
- Spectral energy at certain frequency bands are preserved

MPEG

- Sort-of like lots of JPEGs
- Same 16x16 px macroblock → 6 8x8 px block structure
- Same DCT
- Three types of frame: I-frame, P-frame and B-frame
- Blocks can either be stored completely or store the differences from the previous frame
- Motion compensation: store the location of the most similar block in the previous frame

MPEG



I have a thing for corrupt women.

To the future...

- New audio codec: OPUS
  - Combines Skype's speech codec with “CELT”
  - IETF standard, very good a lower bitrates
- The next barrage of Video codecs:
  - HEVC (h.265): Almost done, bascially h.264 but fancier
  - VP9: WebM but fancier
  - Daala: Xiph.org next-gen video codec with overlapping transforms (still in the planning stage)

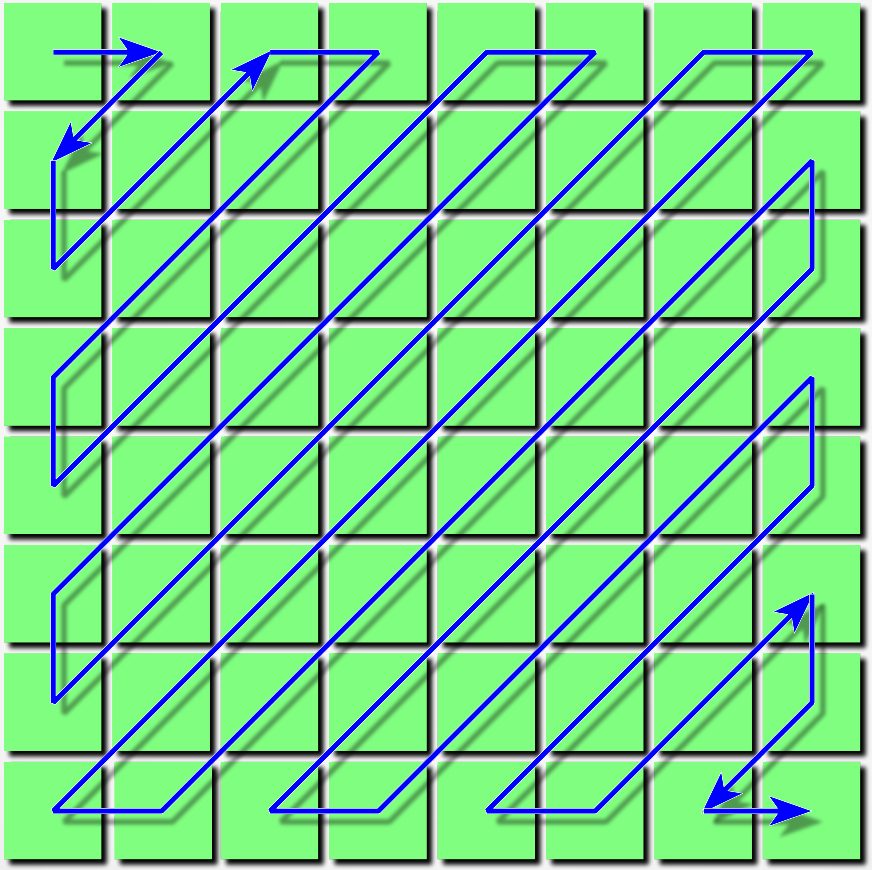
The End

Questions and Chit-chat

Arithmetic coding

- Like Huffman
- Instead of a simple binary tree, you have a weighted n-ary tree.
- Convert an entire stream into a single number
- For each incoming symbol:
  - Split the current range into different sized intervals
  - e.g. [0-0.25] = 0, [0.25-1] = 1
  - Then recurse: [0-0.2] = 00, [0.2-0.25] = 01
  - Then just store a decimal within the correct region for the file
- More efficient than Huffman: theoretically ideal etropy coder
- Patents!

JPEG Zigzagging



h.264 / MPEG4 AVC

- Like MPEG
- In-loop deblocking filter
- Support for 4x4 transforms
- Uses a custom integer HCT (h.264 cosine transform)
  - Plus a Hadamard transform for DC
- Sub-pixel motion prediction
- 10-bit channels (Still new)