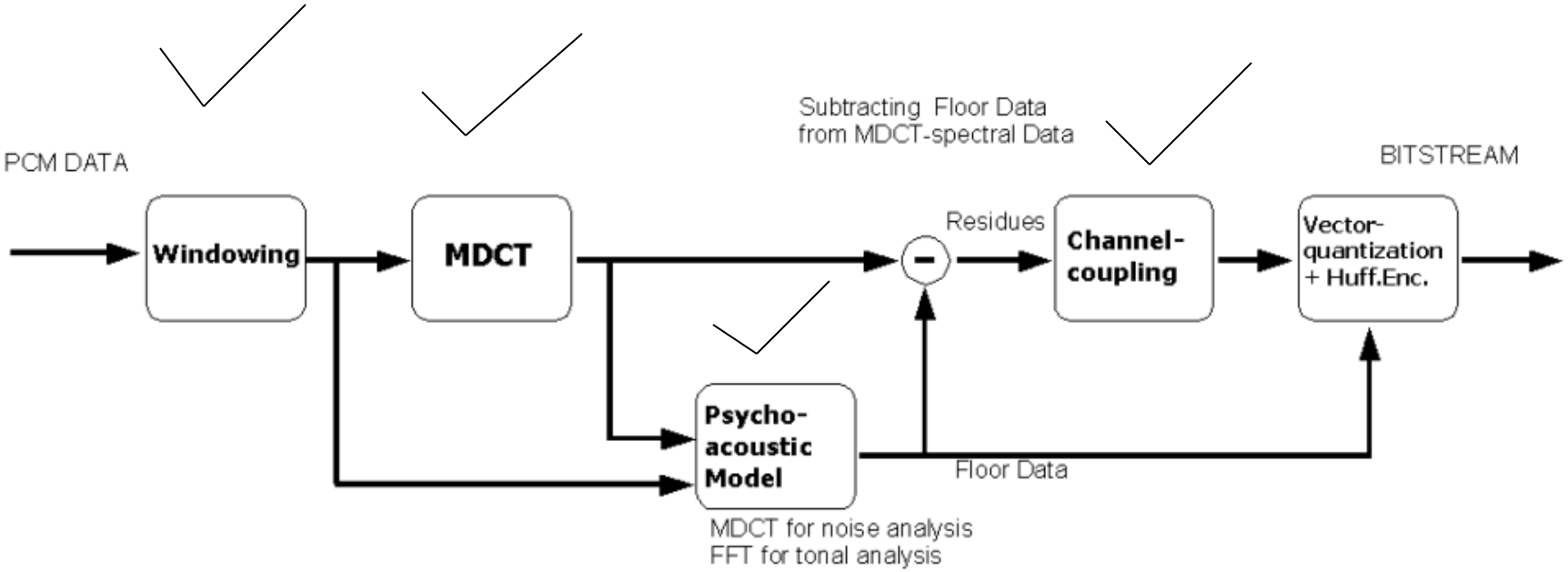
Vorbis Codec (contd.) and Ogg Specifications and Decoding

By

Bharan, Kulin, Prasad and Sanjay

Vorbis audio codec



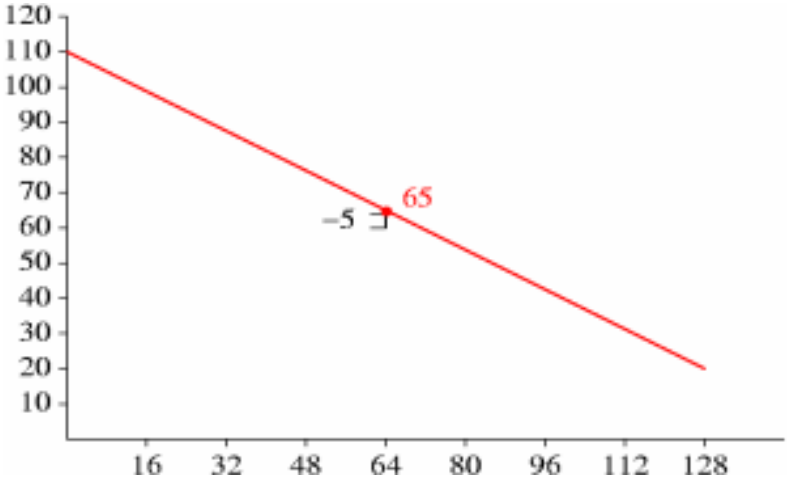
Floor

- Low-resolution parts of audio spectrum
- Number of floor types used and configurations specified in setup header
- Floor type 0
  - Floor 0 uses a packed LSP representation on a dB amplitude scale and Bark frequency scale
  - Not known to be used except by beta4 of the reference encoder
- Floor type 1
  - Piecewise linear interpolated representation on a dB amplitude scale and linear frequency scale
  - Easier to decode than type 0

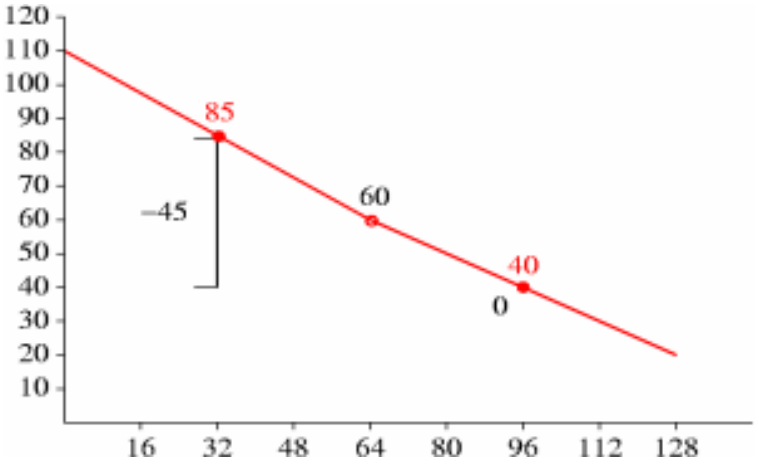
Floor I

- Floor type I → represents a spectral curve as a series of line segments
- Example : The list of selected X values in increasing order is 0,16,32,48,64,80,96,112 and 128. In list order, the values interleave as 0, 128, 64, 32, 96, 16, 48, 80 and 112. The corresponding list-order Y values as decoded from an example packet are 110, 20, -5, -45, 0, -25, -10, 30 and -10. We compute the floor in the following way

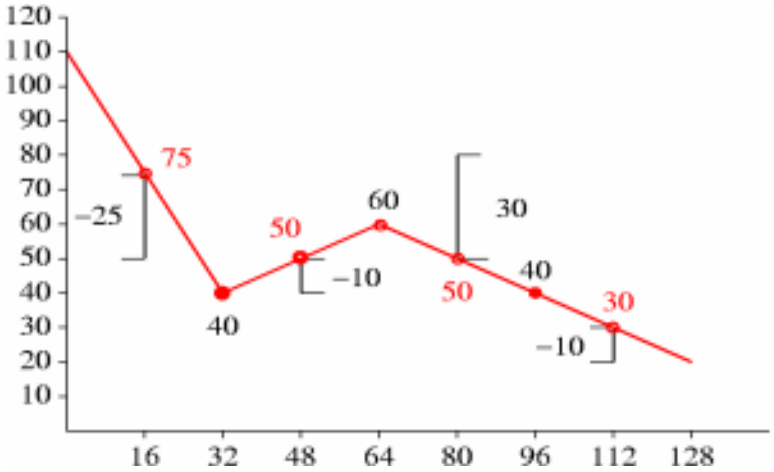
Beginning with the first line



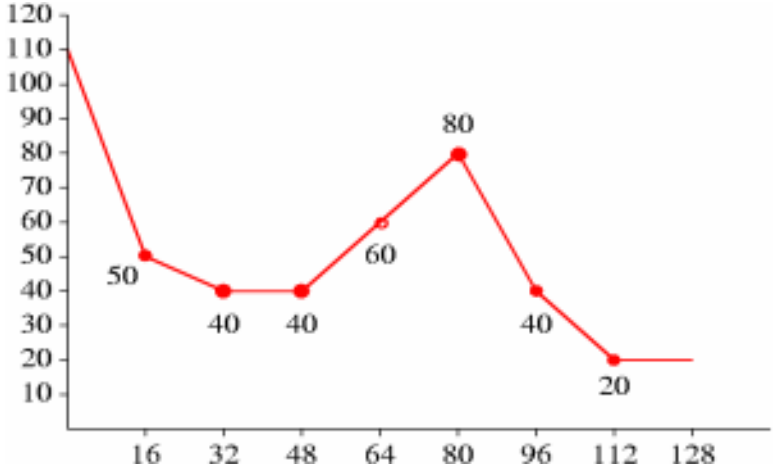
corrected Y at x =64, iterate for X = 32 &96



New_Y for different X



Complete floor pattern

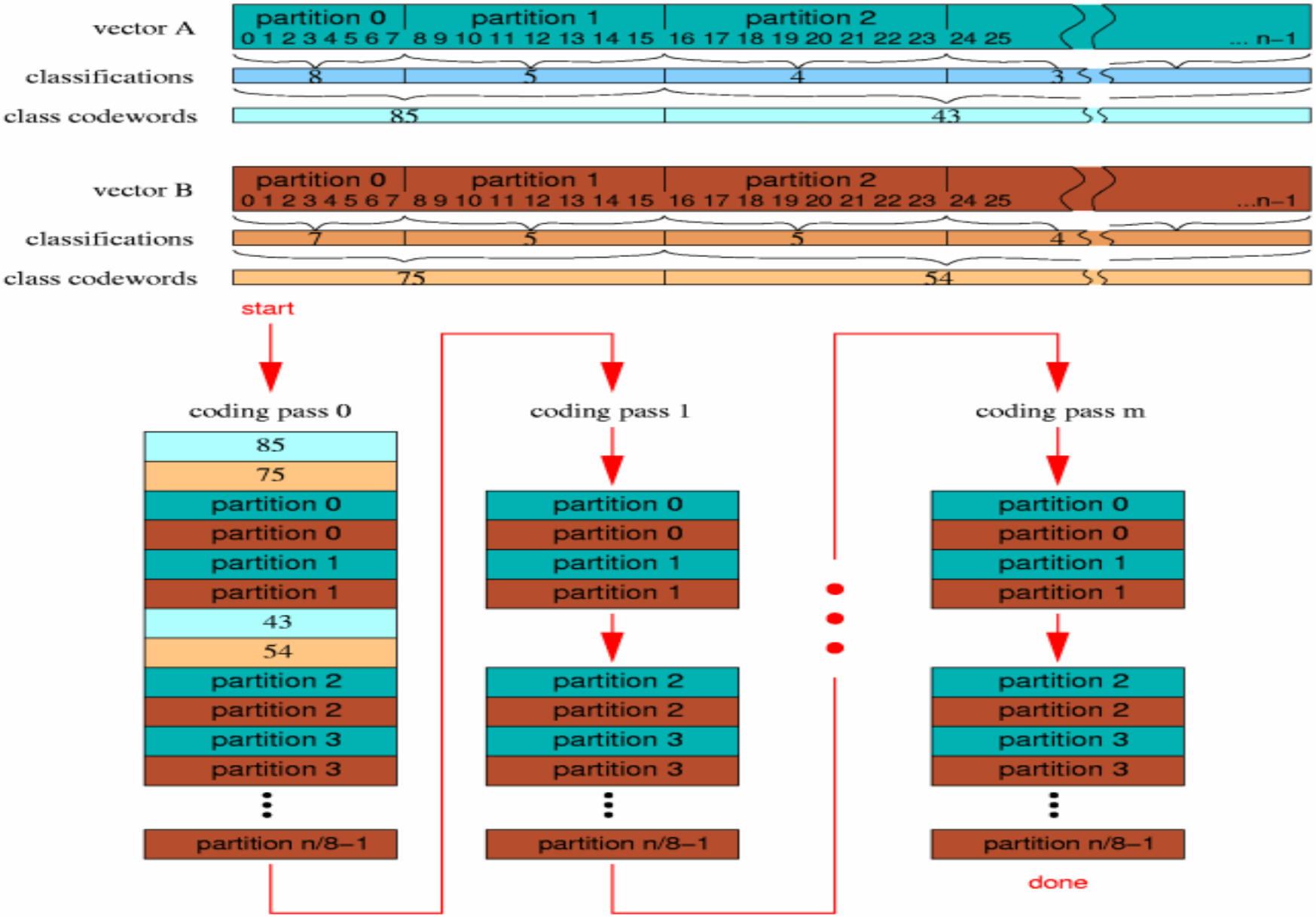


Residue

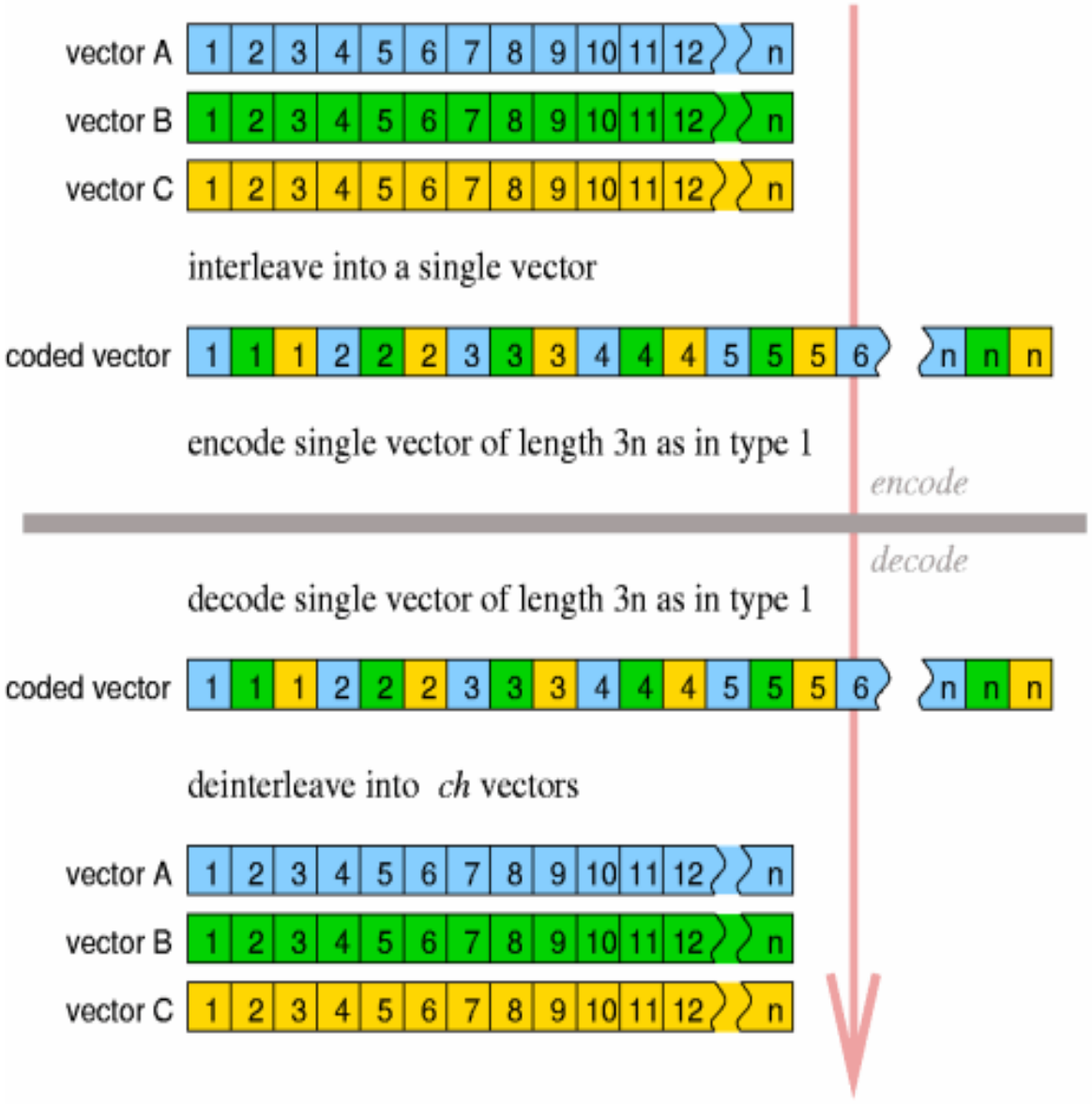
- Residue = Spectrum - Floor
- Number of residue types used and configurations stored in setup header
- Type 0: VQ encoding is interlaced
- Type 1: VQ encoding is not interlaced
- Type 2: Several vectors are interlaced and then flattened

Residue 0

vectors = 2
vector partition size =8
possible classifications = 10
classifications per classification codeword = 2



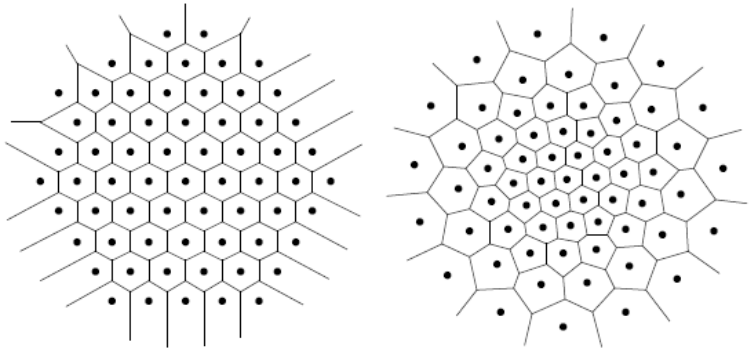
Residue 2


vector A 1 2 3 4 5 6 7 8 9 10 11 12 n
vector B 1 2 3 4 5 6 7 8 9 10 11 12 n
vector C 1 2 3 4 5 6 7 8 9 10 11 12 n
interleave into a single vector
coded vector 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 n n n
encode single vector of length 3n as in type 1
encode
decode
decode single vector of length 3n as in type 1
coded vector 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 n n n
deinterleave into ch vectors
vector A 1 2 3 4 5 6 7 8 9 10 11 12 n
vector B 1 2 3 4 5 6 7 8 9 10 11 12 n
vector C 1 2 3 4 5 6 7 8 9 10 11 12 n

Vector quantization

- Lookup type
  - Type 0: no lookup
  - Type 1: lattice VQ (mostly used)
  - Type 2: tessellated VQ
- No shared codebooks
  - VQ codebooks and Huffman codes computed as part of encoding
  - Codebooks stored at start of bitstream

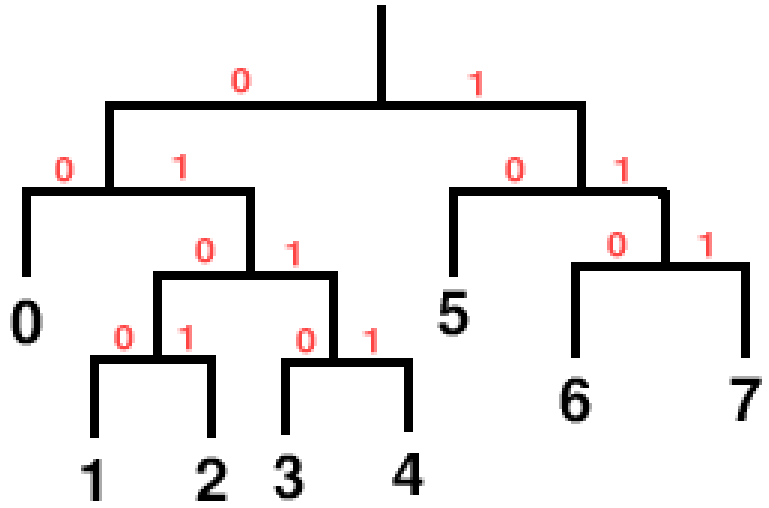
Lattice quantization



- Lattice VQ is a constrained VQ technique, where the code vectors form a highly regular structure
- The regular structure makes compact storage
- Fast *nearest-neighbor search* (finding the closest code vector to an input vector)

Huffman entropy encoding

- The entropy coding in a Vorbis I codebook is provided by a standard Huffman binary tree representation



Mappings

- Holds a channel coupling description
- Mappings can be used to determine which vectors are used to decode which audio channels
- Comprised of submaps
- Each submap corresponds to a subset of the floor and residue vector configurations

Modes

- Master arm switch for packet coding
  - A specific mode number identifies a specific *configuration*
    - Transform used
    - Frame size
    - Mapping number
  - This allows us to tweak how individual frames are coded
- Mode configurations stored in setup header

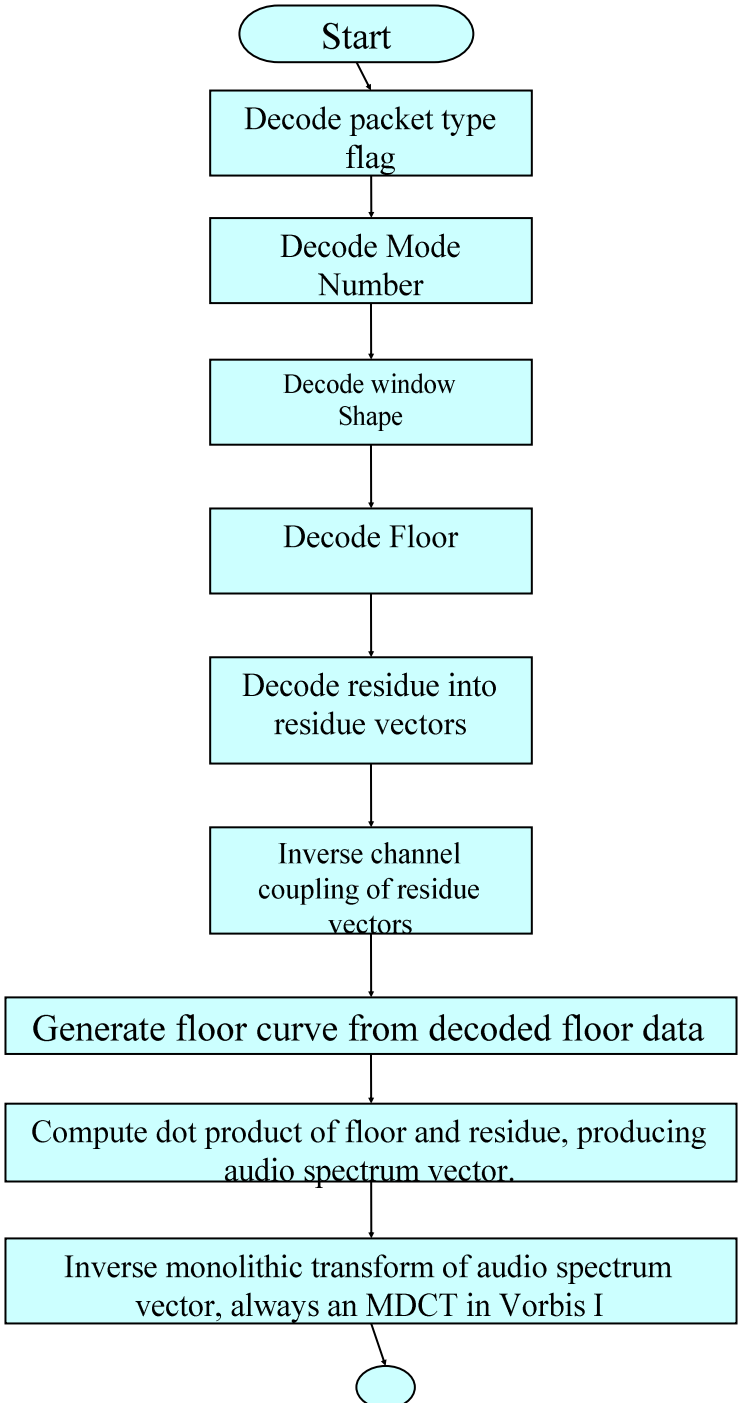
Header packets

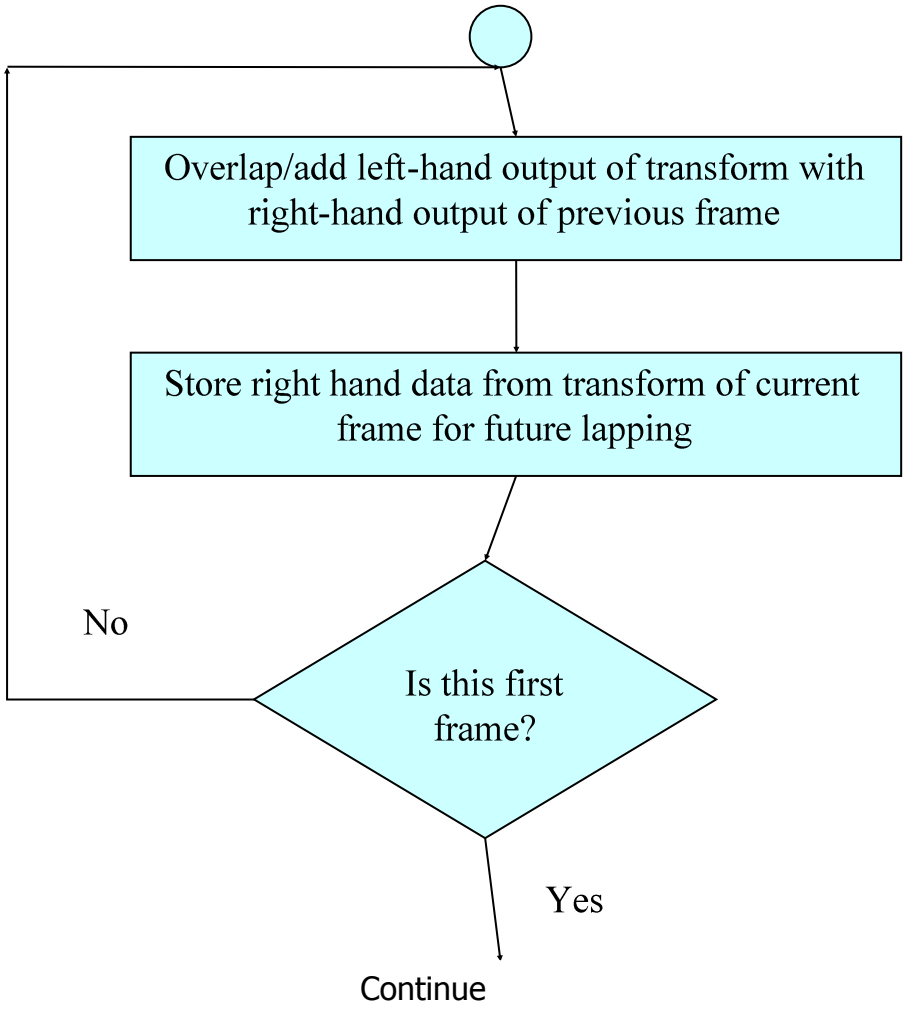
- Identification header → Vorbis version number, Number of channels, Sample Rate
- Comment header → Title, Version, Artist, Genre, Date, Copyright, etc.
- Setup Header → The setup header contains, in order, the lists of codebook configurations, time-domain transform configurations, floor configurations, residue configurations, channel mapping configurations and mode configurations

Audio packets

- Type flag
- Mode number
- Window shape
- Floor (ordered by channel)
- Residue (ordered by submap)

Vorbis decoding





Ogg encapsulation

- Octet vectors of raw, compressed data or packets - no boundary info
- To provide framing/sync, sync recapture after error, landmarks during seeking, and enough info to separate data back into packets at orig. packet boundaries, without relying on decoding to find the packet boundaries

Ogg encapsulation

- Logical and physical
- Decode single pages at a time from the overall bitstream
- Logical – contiguous stream of seq. pages belonging to a single codec instance
- Physical – composed of multi. Logical bitstreams – simplest is the degenerate bitstream
- Example: Vorbis audio, Tarken Video

Design constraints

- True streaming
- No more than 1-2% of bitstream bandwidth
- Abs. position within original stream
- Limited editing, simplified concatenation mechanism
- Detect corruption, frame sync, random access to data at arbitrary positions

Bitstream structure

- Packet segmentation – lacing values.
- Packet can span multiple pages, though individual segments cannot
- Avoids a max. packet size
- Nominal page size of 4-8 kB recommended
- Encoding looks odd, but is properly optimized for speed – various sizes can be handled

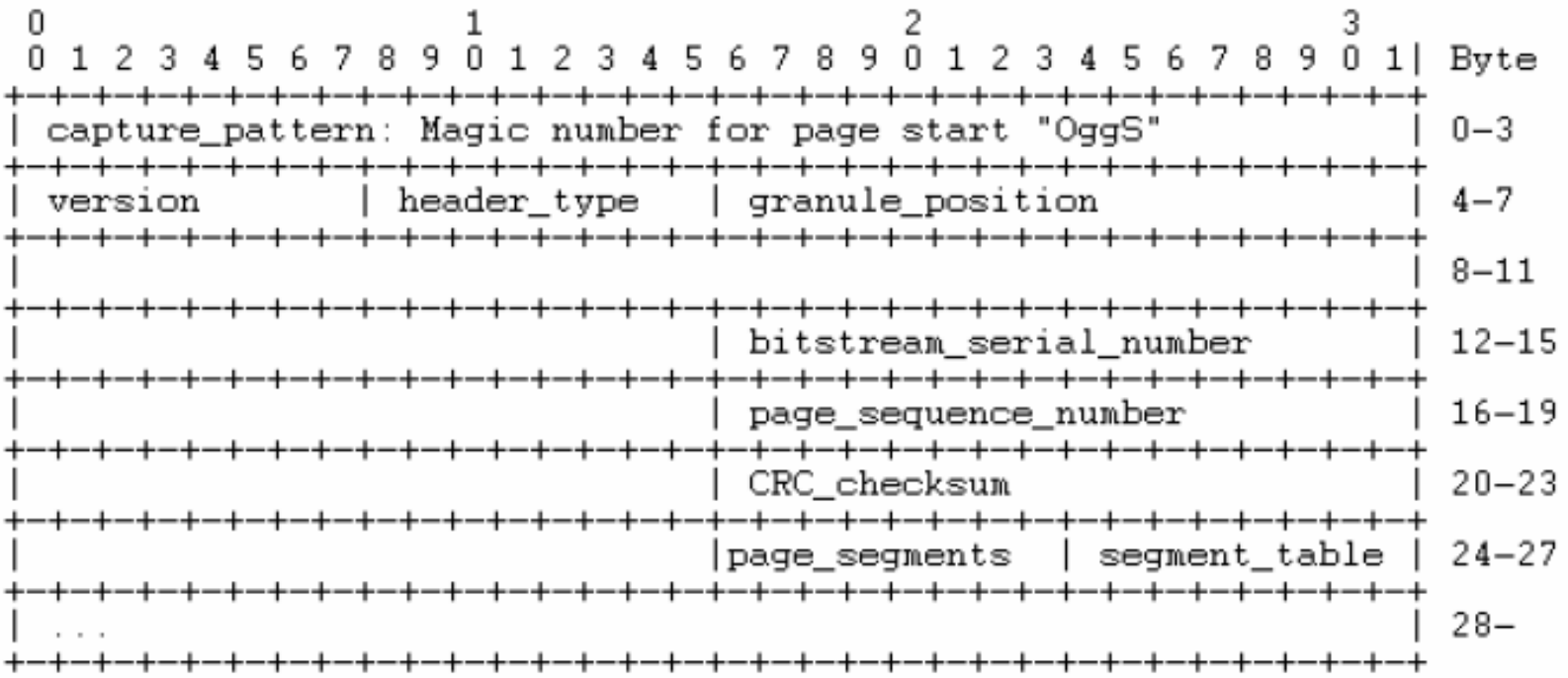
Page format

- Capture pattern (0-3)
- Stream structure version (4)
- Header type flag (5)
- Absolute granular position (6-13) – little endian – a truncated stream will also work (wow!)
- Stream serial no. (14-17)
- Page sequence no. (18-21)

Page format

- 32-bit CRC checksum (22-25)
  - direct algorithm, initial value and final XOR = 0, generator polynomial=0x04c11db7
- Page_segments (26)
- Segment_table (containing the packet lacing values)

Page format

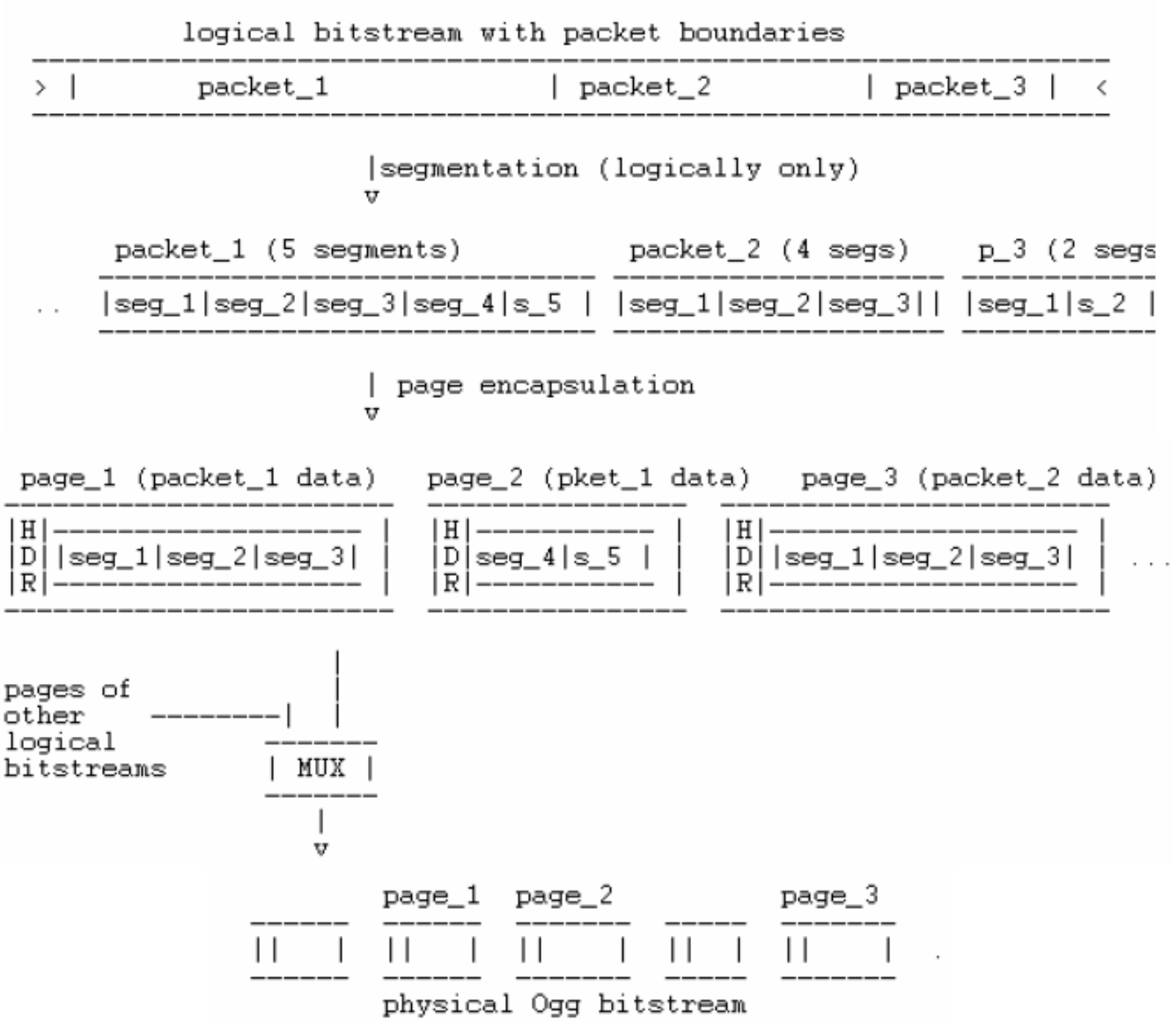

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Byte
capture_pattern: Magic number for page start "OggS"
0-3
version
header_type
granule_position
4-7
8-11
bitstream_serial_number
12-15
page_sequence_number
16-19
CRC_checksum
20-23
page_segments
segment_table
24-27
...
28-

Additional end-to-end structure

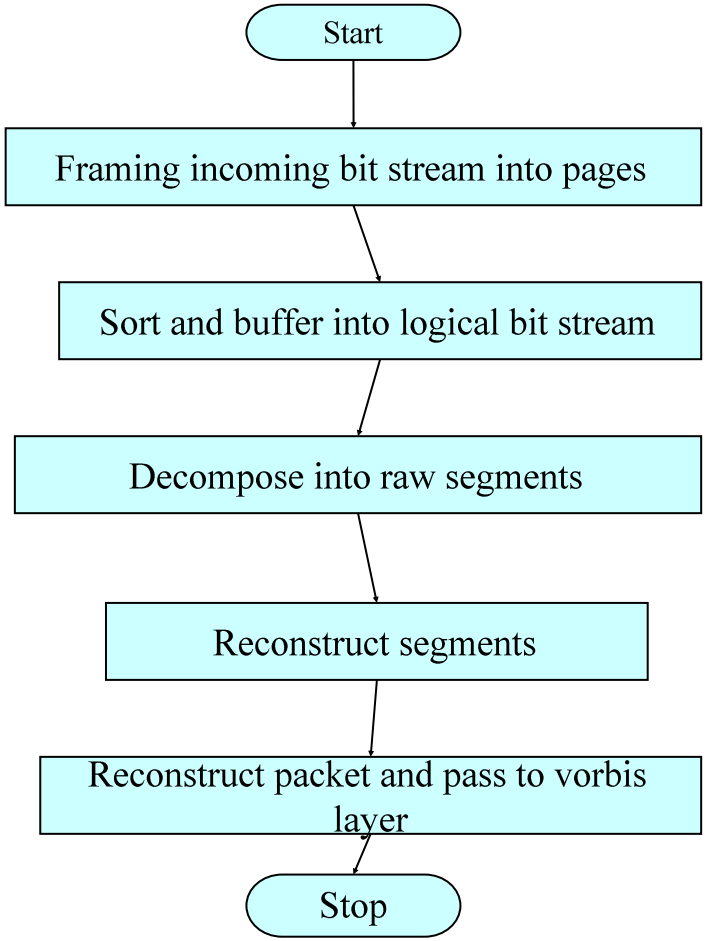
- BOS
- EOS
- First page – contains initial header packet – codec type recognition
  - Vorbis places name and rev. of the codec, audio rate and audio quality

Ogg multiplexing

- Chaining (sequential) – used in Ogg Vorbis
- Concurrent (grouped)
  - Initial pages must appear together
- Mixed (combo of the prev. two)


logical bitstream with packet boundaries
packet_1
packet_2
packet_3
segmentation (logically only)
packet_1 (5 segments)
seg_1 seg_2 seg_3 seg_4 s_5
packet_2 (4 segs)
seg_1 seg_2 seg_3
p_3 (2 segs
seg_1 s_2
page encapsulation
page_1 (packet_1 data)
H D R
seg_1 seg_2 seg_3
page_2 (pket_1 data)
seg_4 s_5
page_3 (packet_2 data)
seg_1 seg_2 seg_3
pages of other logical bitstreams
MUX
page_1
page_2
page_3
physical Ogg bitstream

Ogg decoding



That’s all folks

Questions?