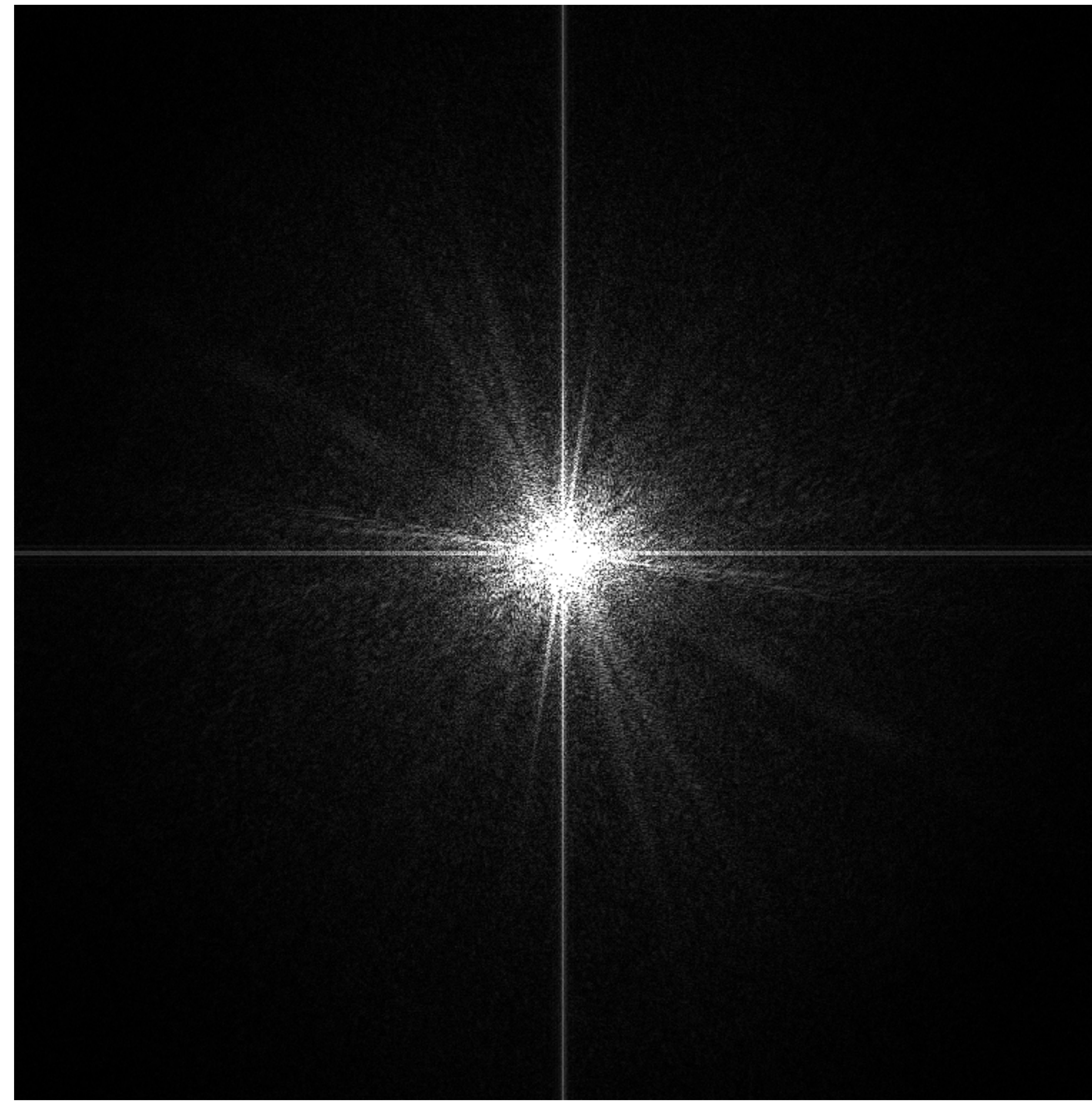
Lecture 6:

Lossy Image and Video Compression

Visual Computing Systems
Stanford CS348V, Winter 2018

Recall: frequency content of images

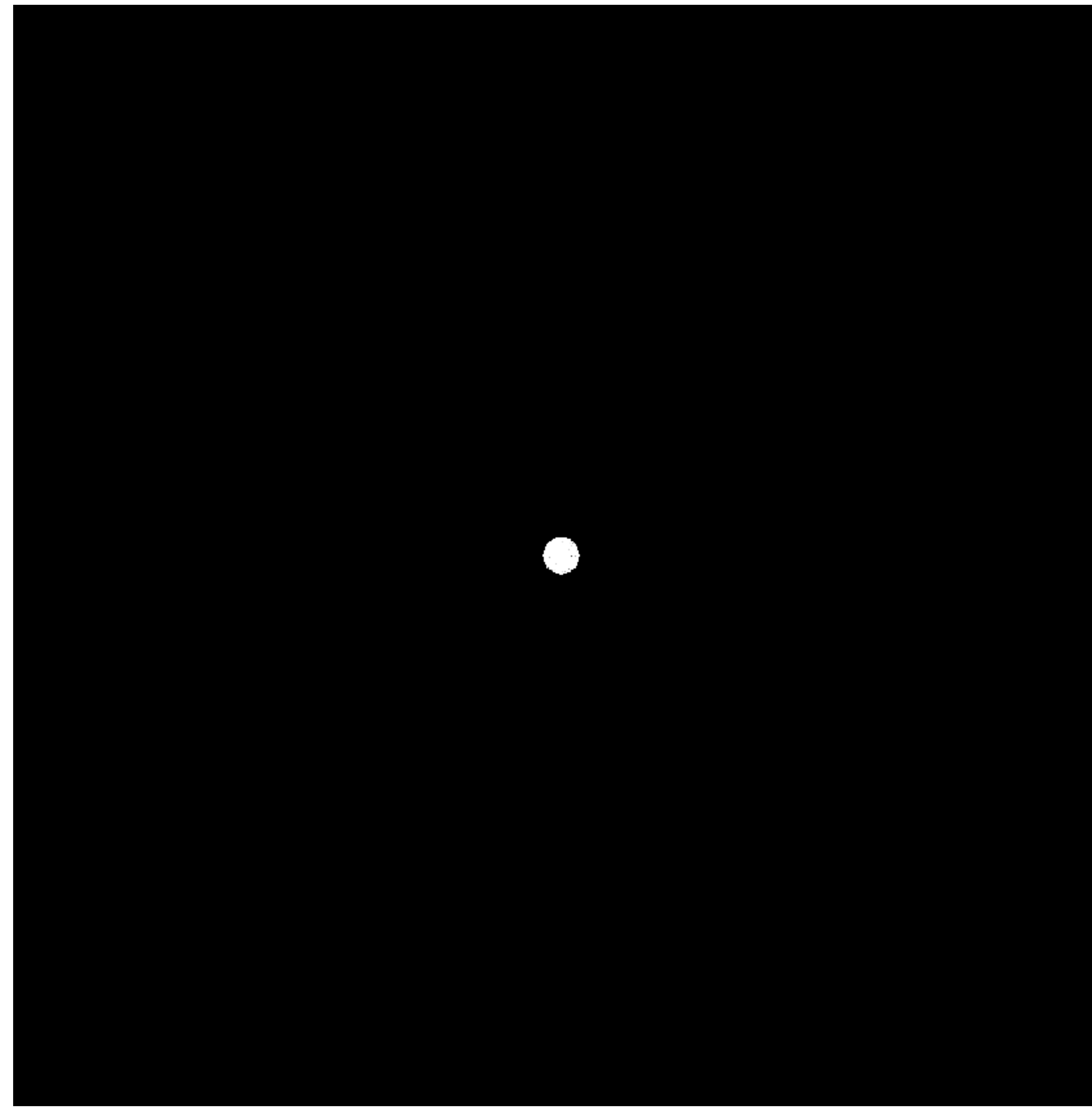
Spatial domain result



Spectrum

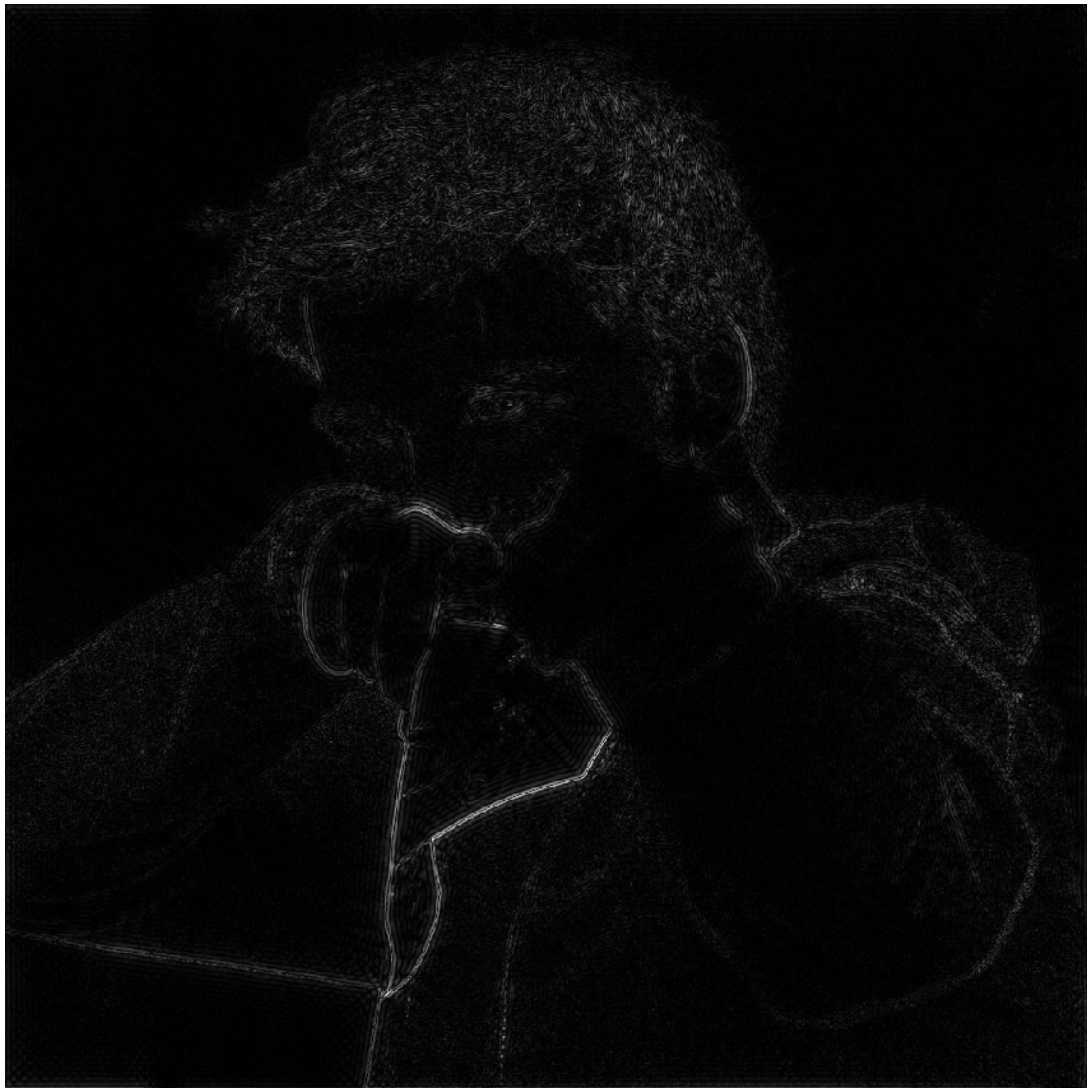
Recall: frequency content of images

Spatial domain result

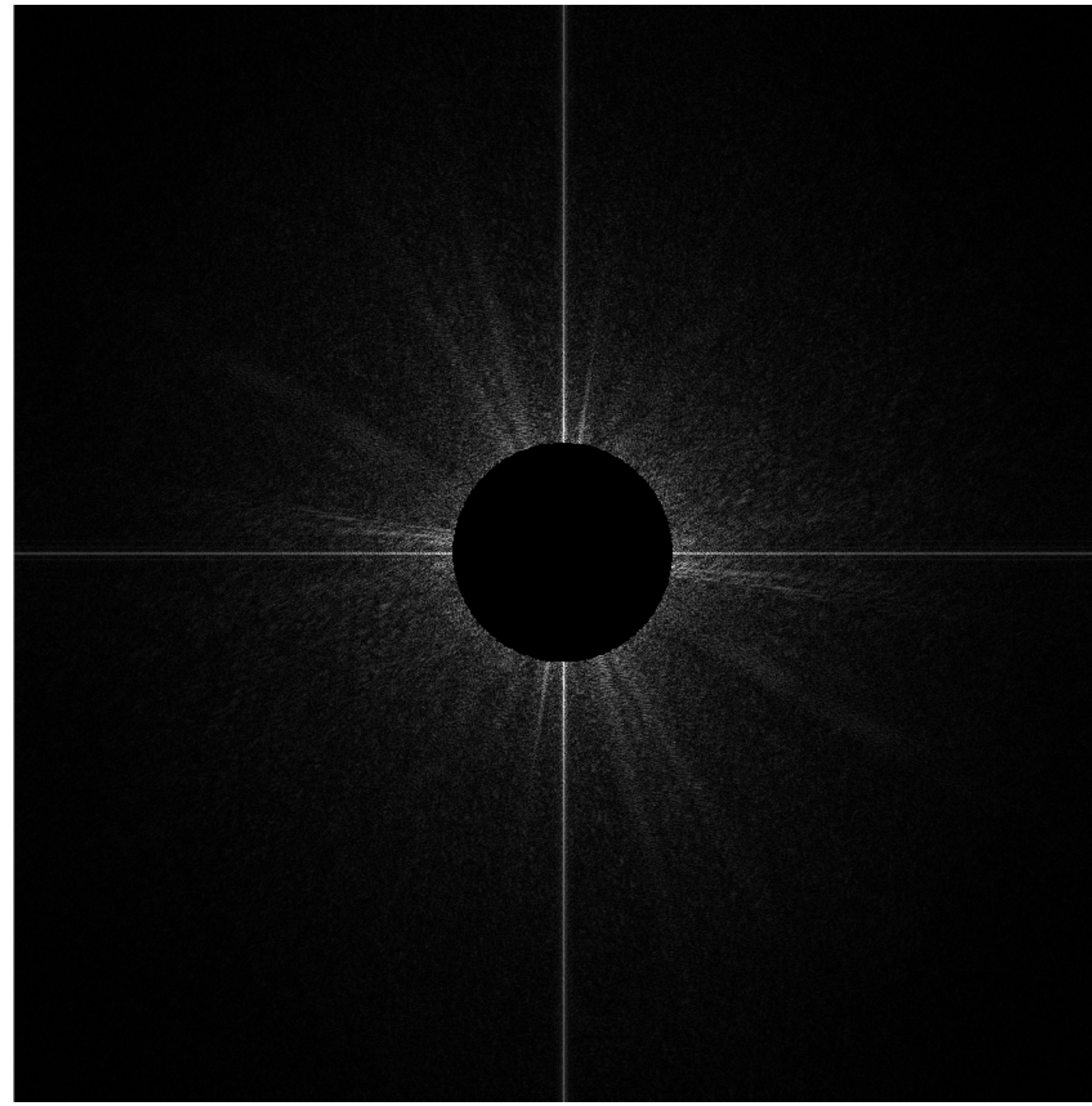


Spectrum (after low-pass filter)
All frequencies above cutoff have 0 magnitude

Recall: frequency content of images



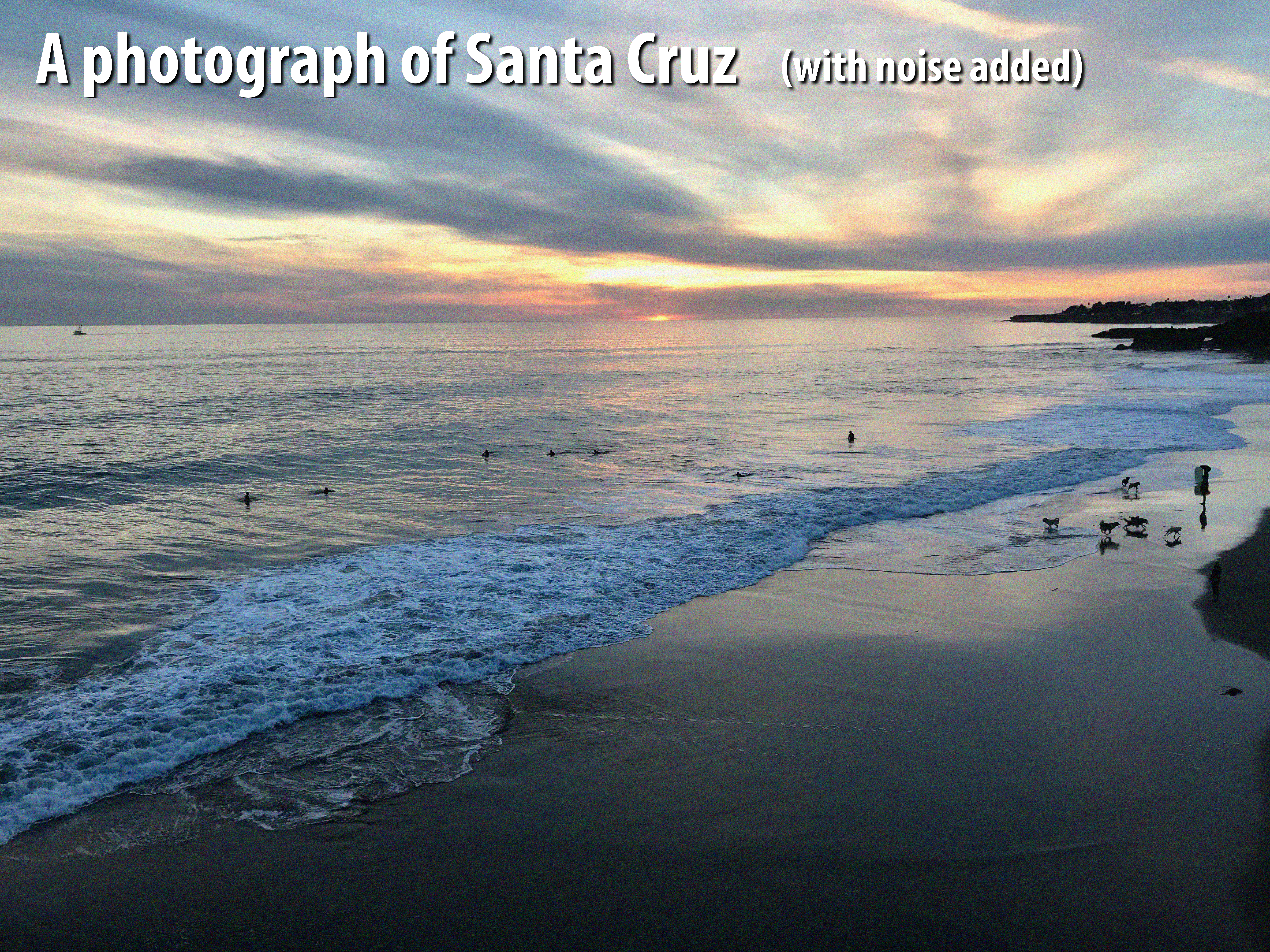
Spatial domain result
(strongest edges)



Spectrum (after high-pass filter)
All frequencies below threshold
have 0 magnitude

A photograph of Santa Cruz

A photograph of Santa Cruz (with noise added)



A photograph of Santa Cruz (with more noise added)

JPEG compression: the big ideas

- Low-frequency content is predominant in images of the real world

Therefore, it is often acceptable for a compression scheme to introduce errors in high-frequency components of the image.

- The human visual system is:
  - less sensitive to high frequency sources of error
  - less sensitive to detail in chromaticity than in luminance

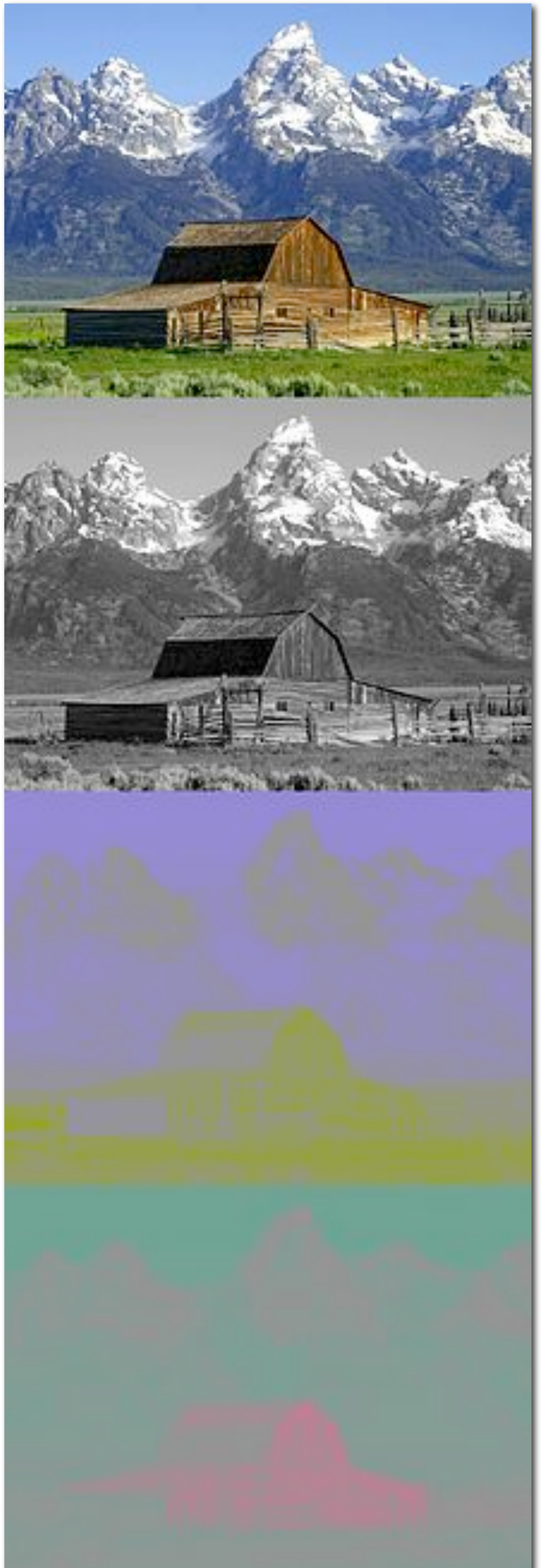
[Credit: Pat Hanrahan]

Y’CbCr color space

Y’ = luma: perceived luminance (non-linear)

Cb = blue-yellow deviation from gray

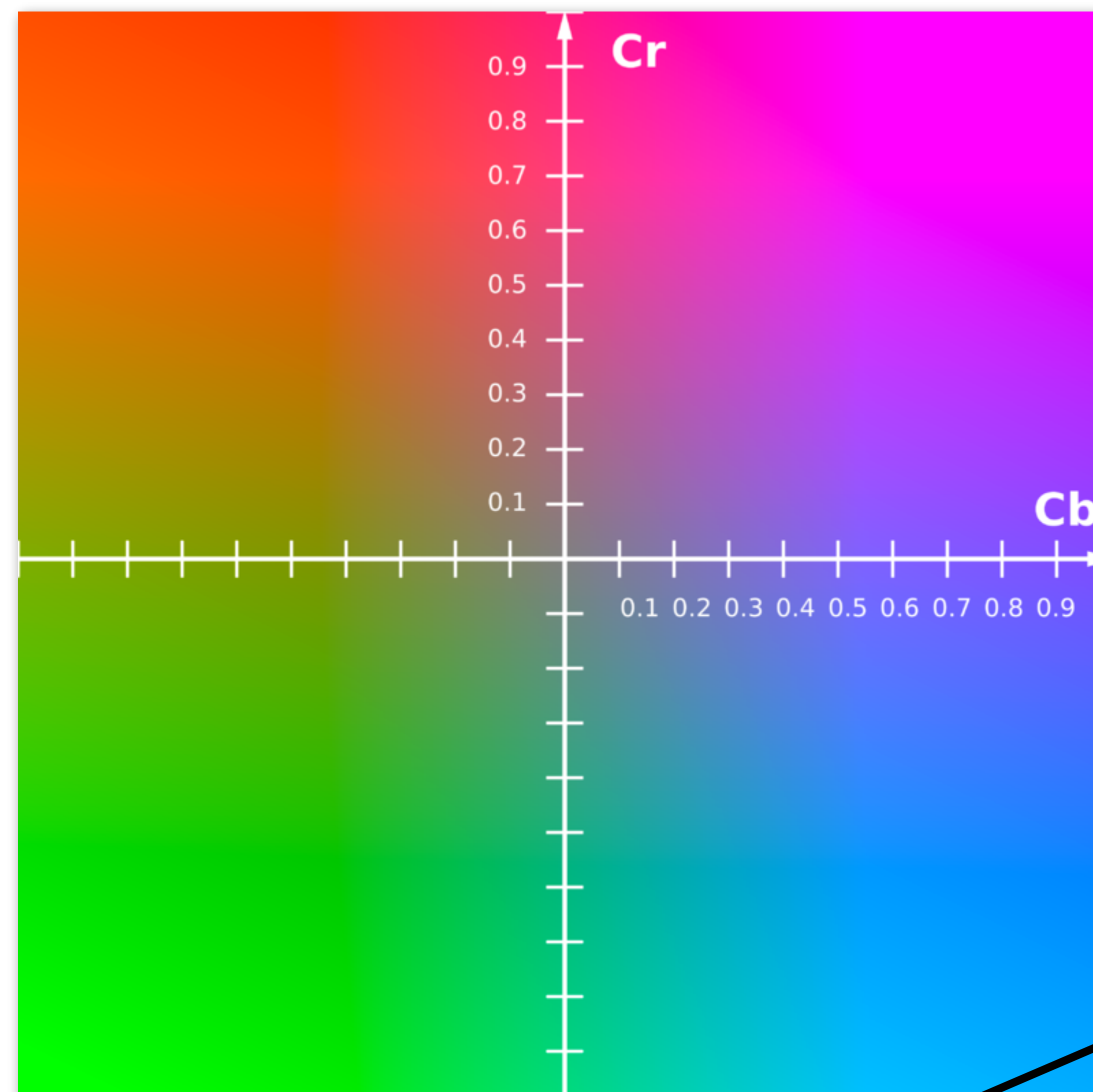
Cr = red-cyan deviation from gray



Y’

Cb

Cr



Non-linear RGB (primed notation indicates perceptual (non-linear) space)

Conversion from R’G’B’ to Y’CbCr:

$$Y' = 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256}$$

$$C_B = 128 + \frac{-37.945 \cdot R'_D}{256} - \frac{74.494 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256}$$

$$C_R = 128 + \frac{112.439 \cdot R'_D}{256} - \frac{94.154 \cdot G'_D}{256} - \frac{18.285 \cdot B'_D}{256}$$

Image credit: Wikipedia

Example: compression in Y'CbCr

Original picture of Kayvon

Example: compression in Y'CbCr

Contents of CbCr color channels downsampled by a factor of 20 in each dimension
(400x reduction in number of samples)

Example: compression in Y'CbCr

Full resolution sampling of luma (Y')

Example: compression in Y'CbCr

Reconstructed result
(looks pretty good)

Chroma subsampling

Y'CbCr is an efficient representation for storage (and transmission) because Y' can be stored at higher resolution than CbCr without significant loss in perceived visual quality

Y'_{00} Cb_{00} Cr_{00}	Y'_{10}	Y'_{20} Cb_{20} Cr_{20}	Y'_{30}
Y'_{01} Cb_{01} Cr_{01}	Y'_{11}	Y'_{21} Cb_{21} Cr_{21}	Y'_{31}

4:2:2 representation:

Store Y' at full resolution

Store Cb, Cr at full vertical resolution, but only half horizontal resolution

Y'_{00} Cb_{00} Cr_{00}	Y'_{10}	Y'_{20} Cb_{20} Cr_{20}	Y'_{30}
Y'_{01}	Y'_{11}	Y'_{21}	Y'_{31}

4:2:0 representation:

Store Y' at full resolution

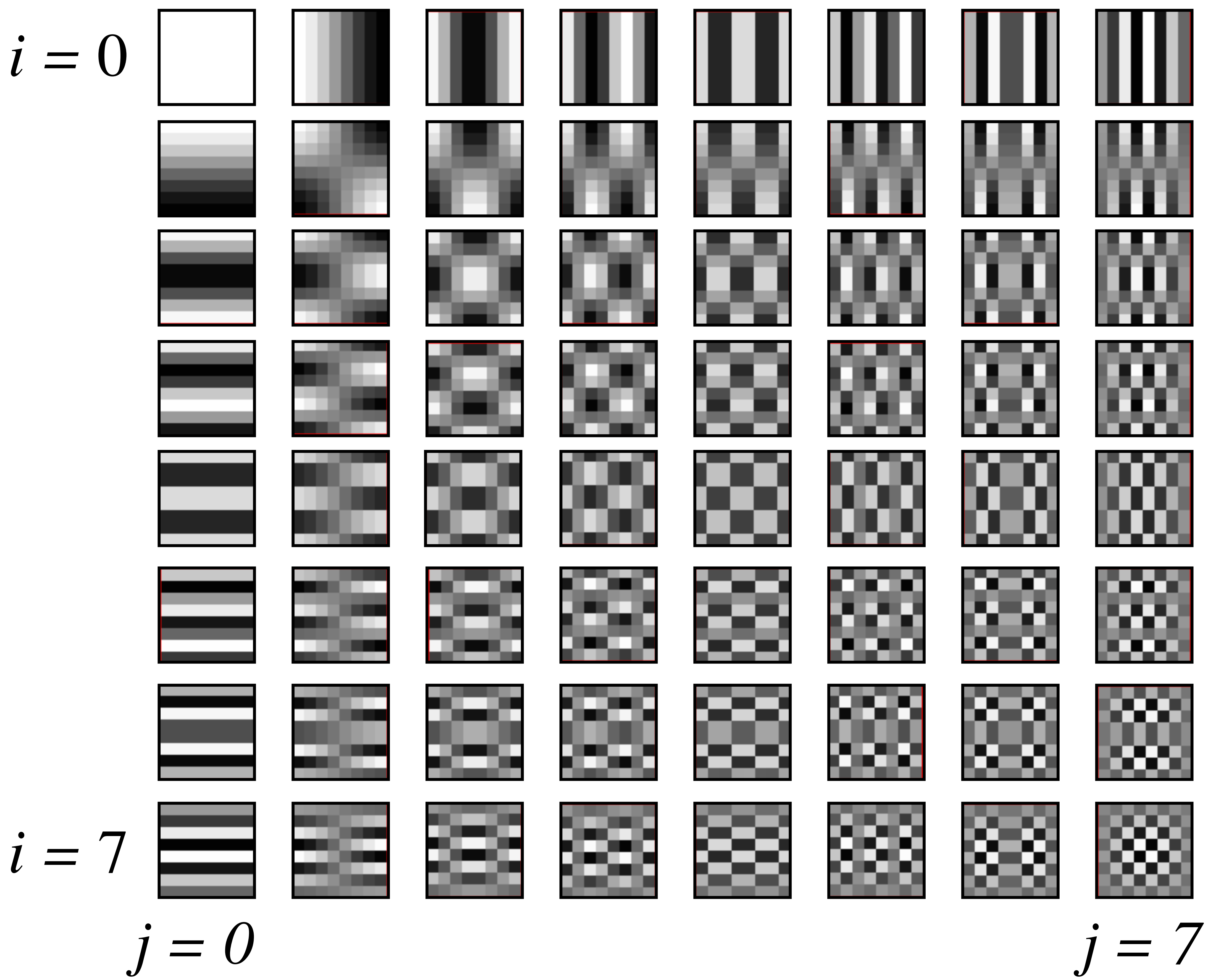
Store Cb, Cr at half resolution in both dimensions

X:Y:Z notation:

- X = width of block
- Y = number of chroma samples in first row
- Z = number of chroma samples in second row

Apply discrete cosine transform (DCT) to each 8x8 block of image values

$$\text{basis}[i,j] = \cos\left[\pi \frac{i}{N}\left(x+\frac{1}{2}\right)\right] \times \cos_{j}\left[\pi \frac{j}{N}\left(y+\frac{1}{2}\right)\right]$$



DCT computes projection of image onto 64 basis functions: basis[i, j]

DCT applied to 8x8 pixel blocks of Y' channel, 16x16 pixel blocks of Cb, Cr (assuming 4:2:0)

Image transform coding via DCT

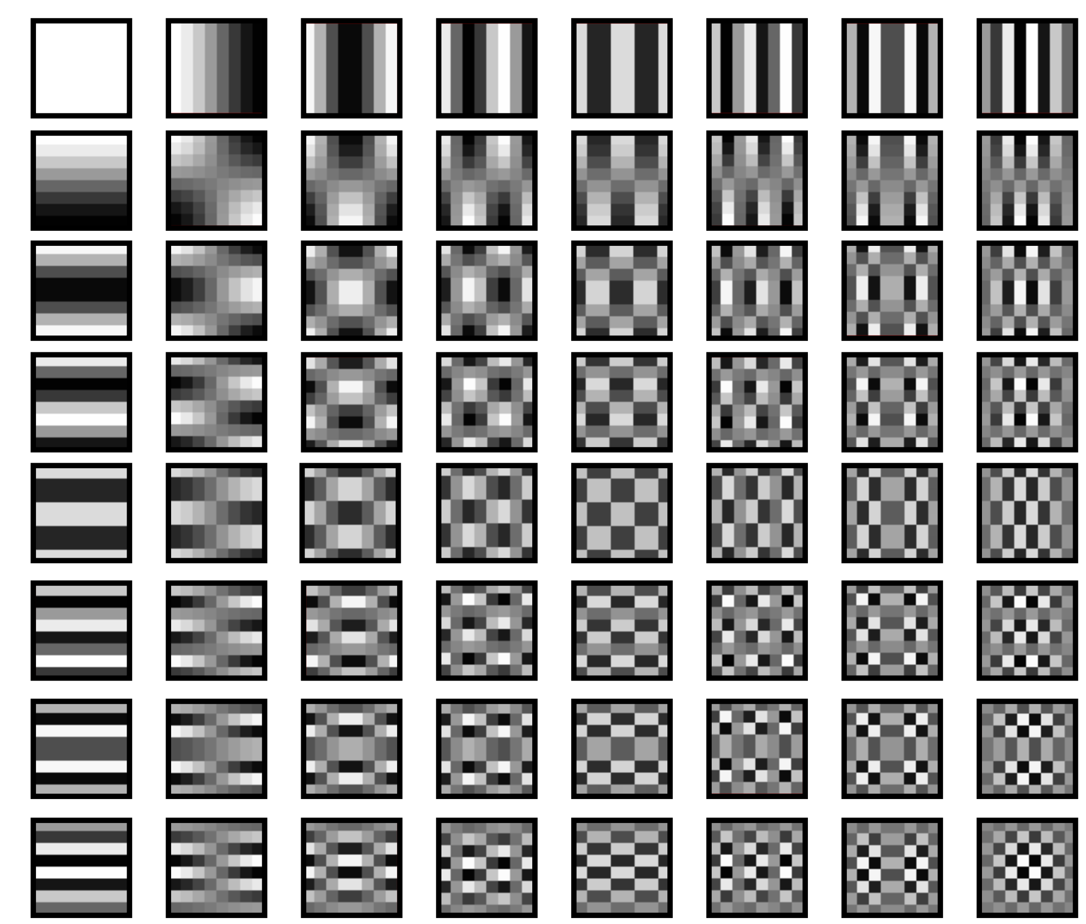
8x8 pixel block
(64 coefficients of signal in "pixel basis")

64 basis coefficients

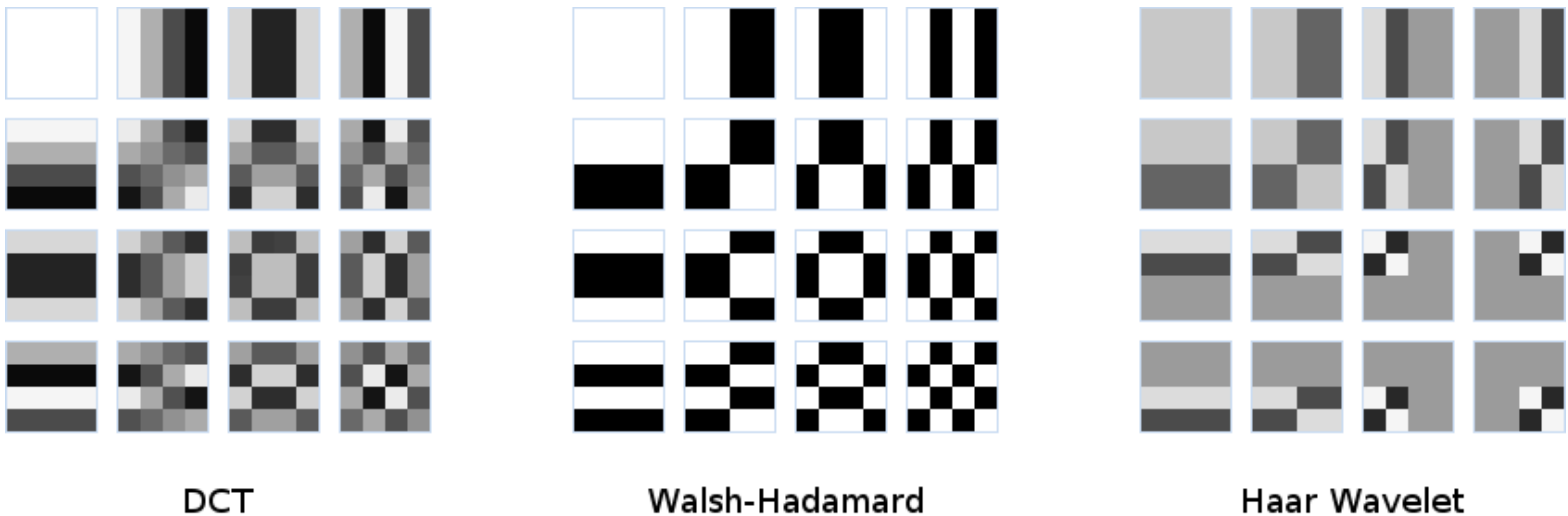
64 cosine basis vectors
(each vector is 8x8 image)

$$\cos\left[\pi \frac{i}{N}\left(x+\frac{1}{2}\right)\right] \times \underset{j}{\cos}\left[\pi \frac{j}{N}\left(y+\frac{1}{2}\right)\right]$$

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



Other bases



This slide illustrates basis images for 4x4 image block

[Image credit: https://people.xiph.org/~xiphmont/demo/daala/demo3.shtml]

Quantization

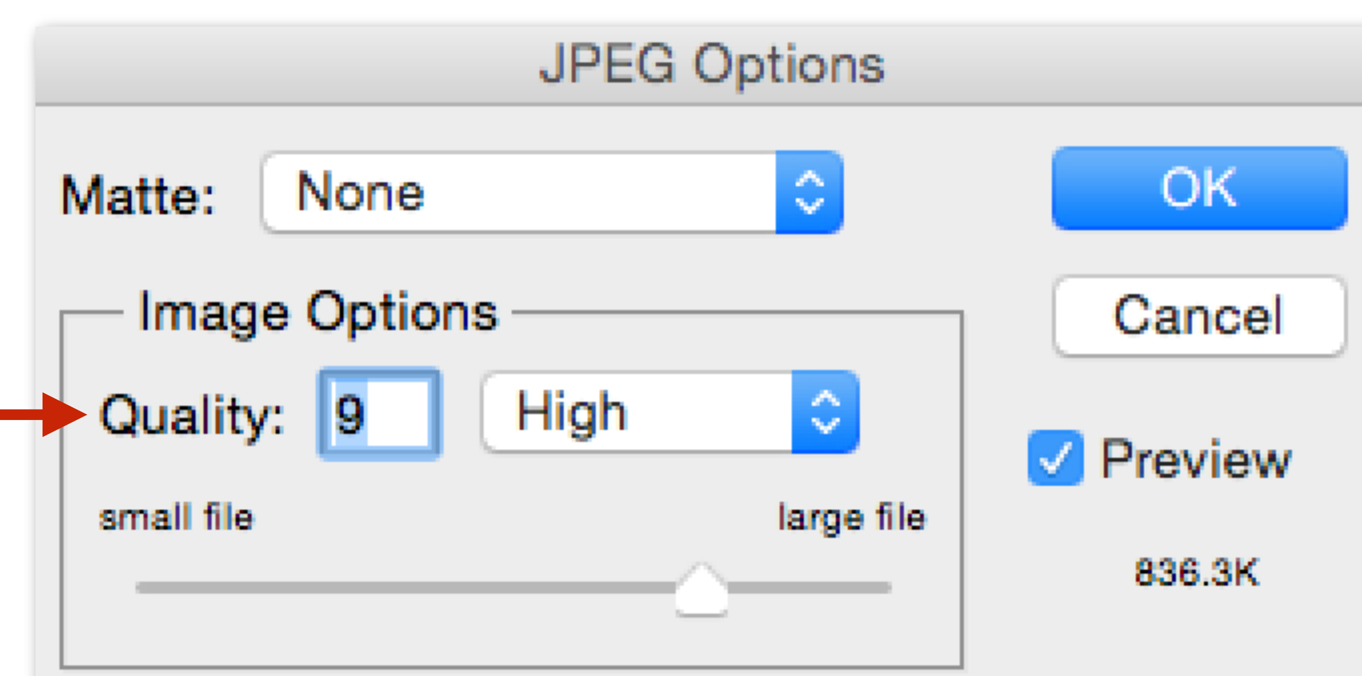
$$\begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix} \Big/ \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Result of DCT
(representation of image in cosine basis)

Quantization Matrix

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

Changing JPEG quality setting in your favorite photo app modifies this matrix ("lower quality" = higher values for elements in quantization matrix)



Quantization produces small values for coefficients (only few bits needed per coefficient)

Quantization zeros out many coefficients

[Credit: Pat Hanrahan]

JPEG compression artifacts

Noticeable 8x8 pixel block boundaries

Noticeable error near large color gradients

Low-frequency regions of image represented accurately even under high compression

JPEG compression artifacts

Original Image

Quality Level 9

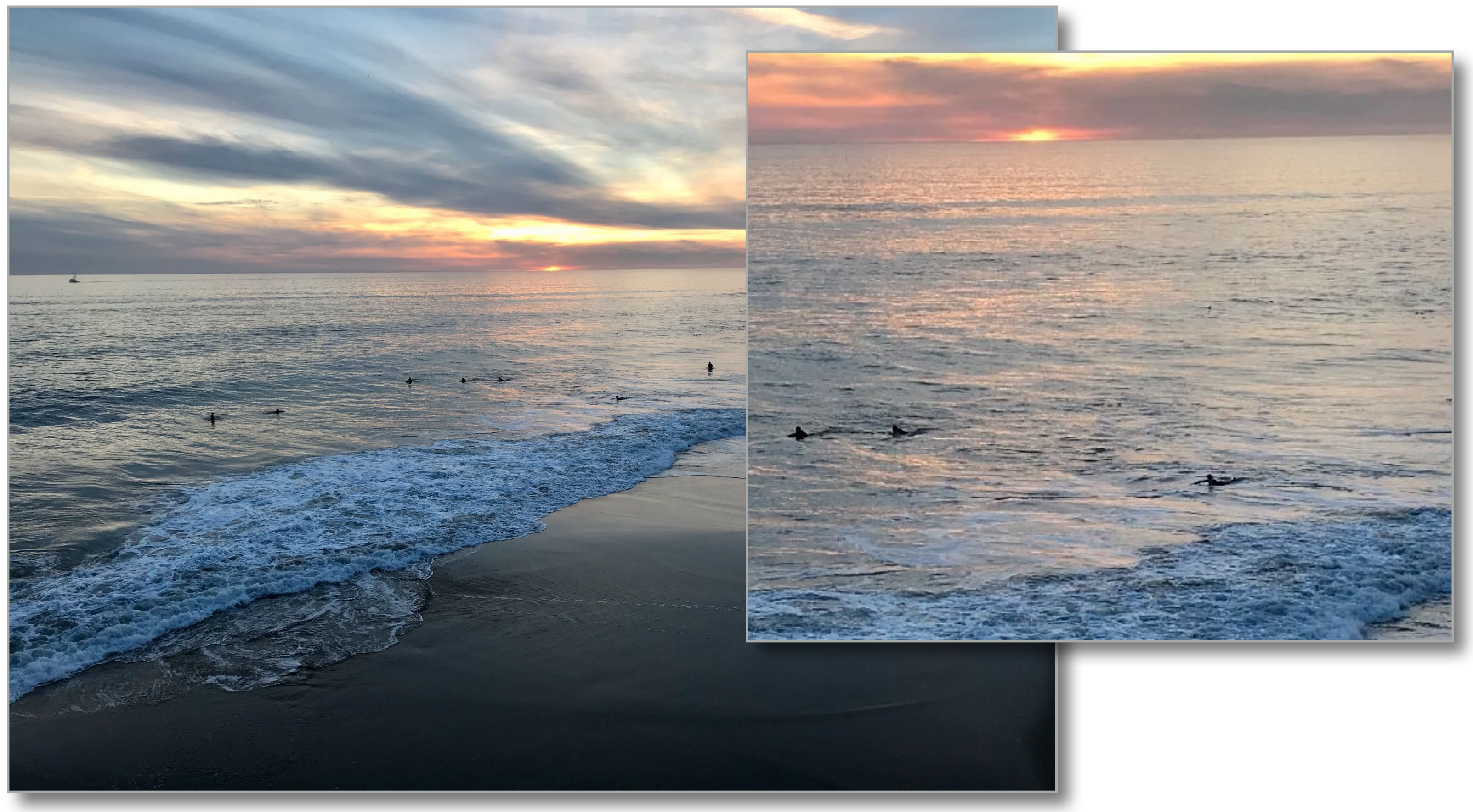
Quality Level 6

Quality Level 3

Quality Level 1

Why might JPEG compression not be a good compression scheme for illustrations and rasterized text?

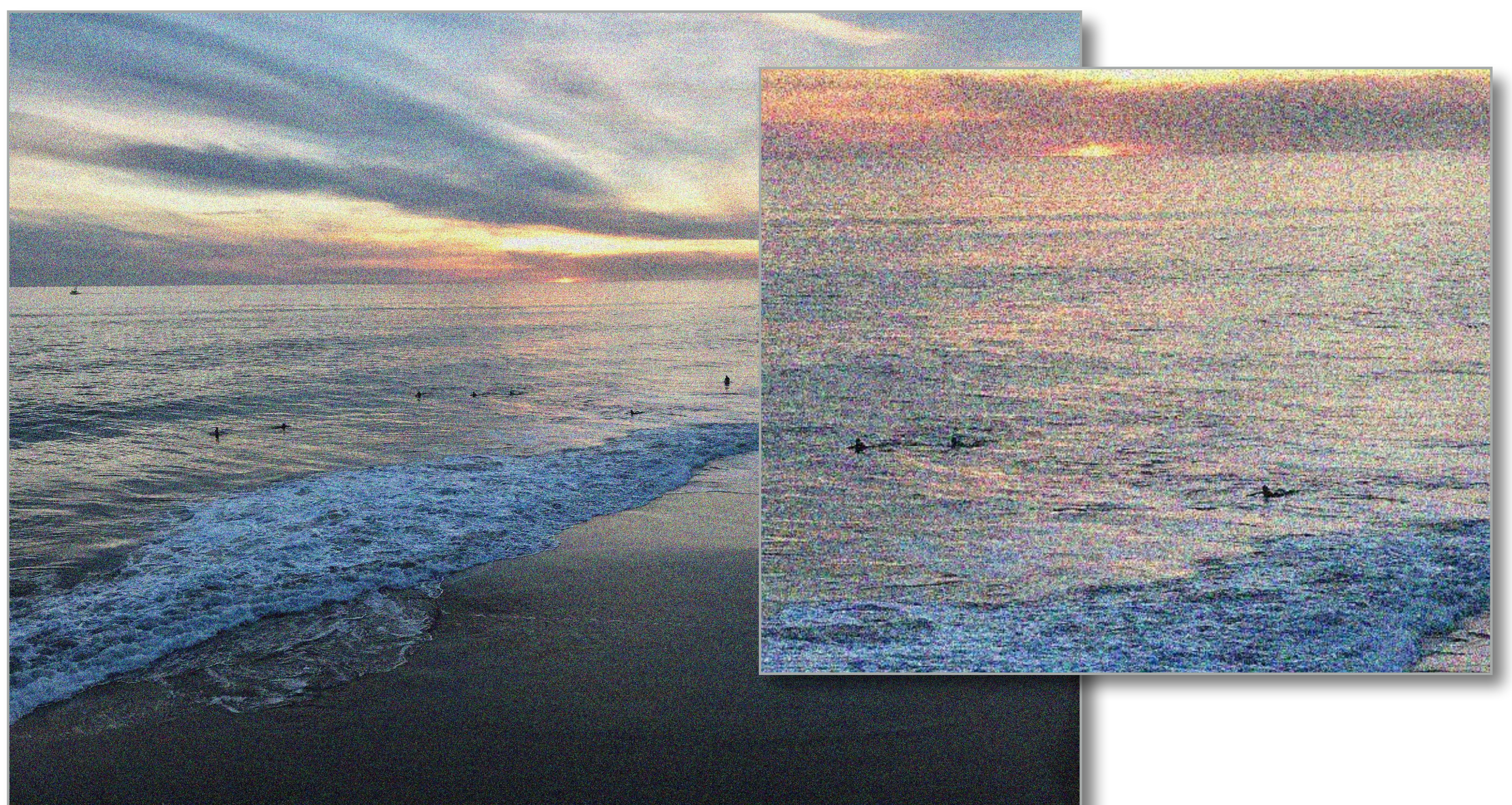
Images with high frequency content do not exhibit high compression ratios. Why?



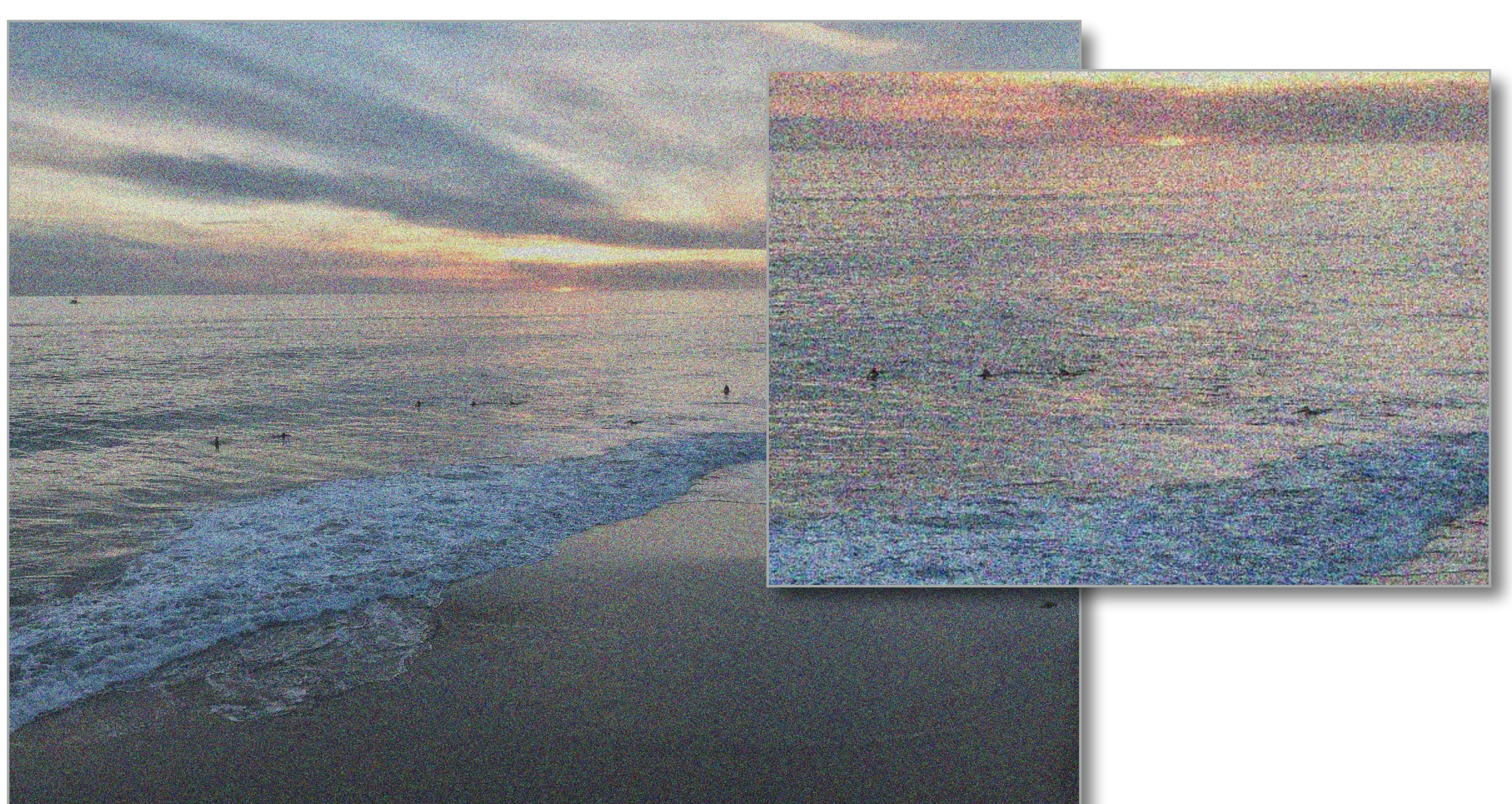
Original image: 2.6MB JPG

Photoshop JPG Compression level = 10 for all compressed images

Uncompressed image:
4032 x 3024 x 24 bbp = 36.6 MB



Medium noise: 20.5MB JPG

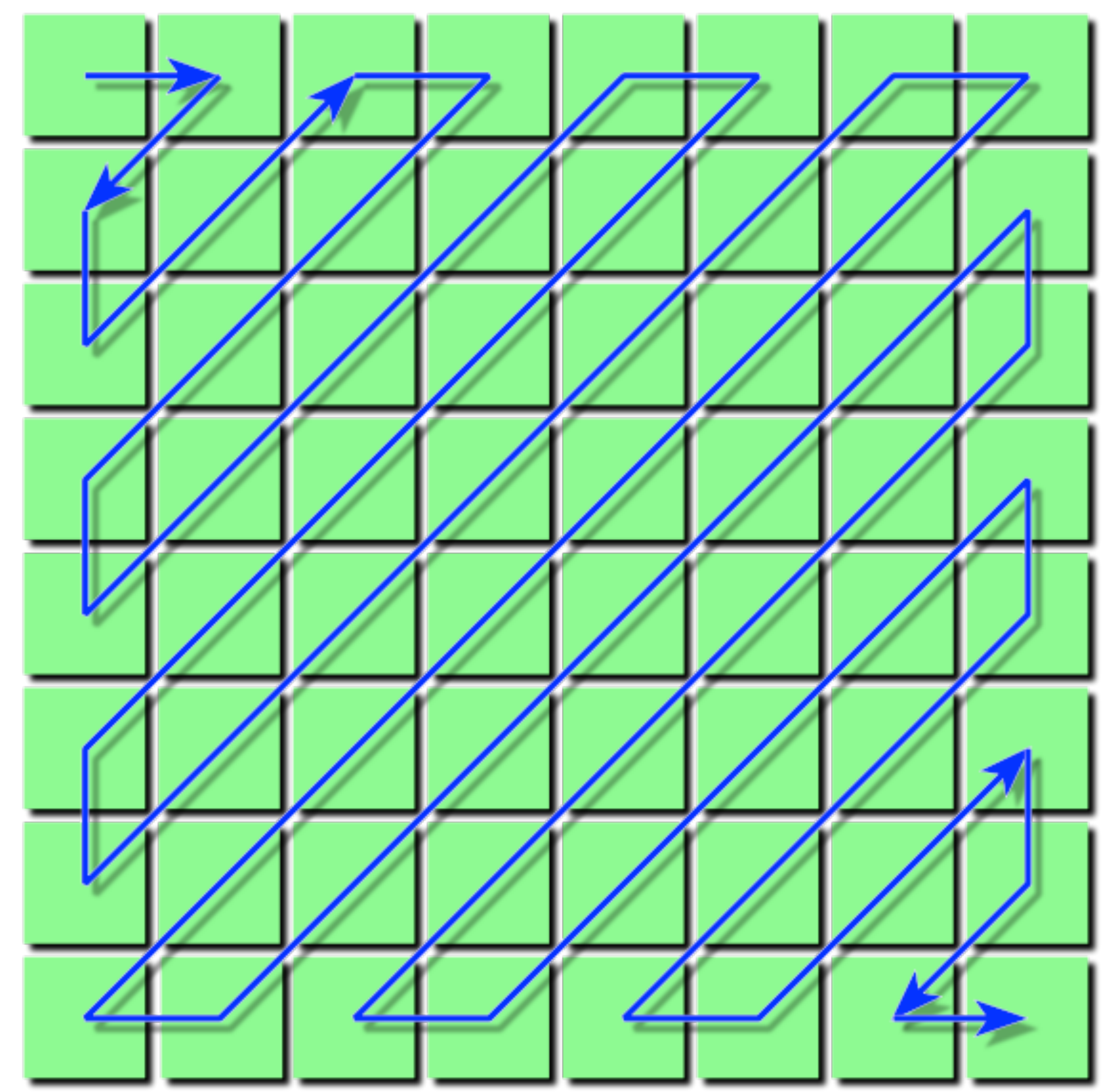


High noise: 26.5MB JPG

Lossless compression of quantized DCT values

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

Quantized DCT Values



Reordering

Entropy encoding: (lossless)

- **Reorder values**
- **Run-length encode (RLE) 0's**
- **Huffman encode non-zero values**

JPEG compression summary

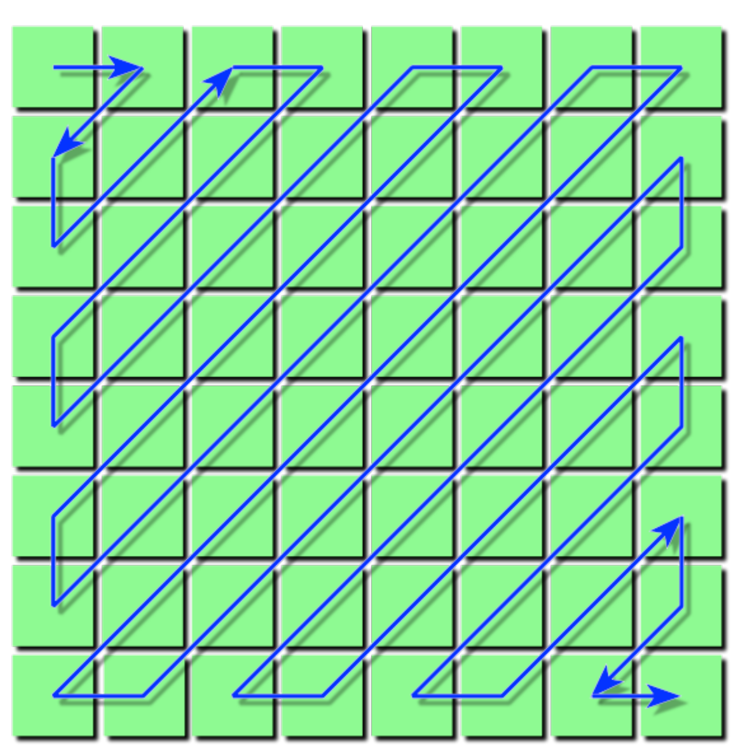
$$
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\
0 & 0 & -1 & -4 & -1 & 0 & 1 & 2
\end{bmatrix}
\Bigg/
\begin{bmatrix}
16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\
12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\
14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\
14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\
18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\
24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\
49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\
72 & 92 & 95 & 98 & 112 & 100 & 103 & 99
\end{bmatrix}
$$

DCT **Quantization Matrix**

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

Quantization loses information (lossy compression!)

Quantized DCT



Coefficient reordering

RLE compression of zeros

Entropy compression of non-zeros

Lossless compression!

Compressed bits

Credit: Pat Hanrahan

JPEG compression summary

Convert image to Y'CbCr

Downsample CbCr (to 4:2:2 or 4:2:0) (information loss occurs here)

For each color channel (Y', Cb, Cr):

- For each 8x8 block of values
  - Compute DCT
  - Quantize results (information loss occurs here)
  - Reorder values
  - Run-length encode 0-spans
  - Huffman encode non-zero values

Key idea: exploit characteristics of human perception to build efficient image storage and image processing systems

- Separation of luminance from chrominance in color representation (Y'CrCb) allows reduced resolution in chrominance channels (4:2:0)
- Encode pixel values linearly in lightness (perceived brightness), not in luminance (distribute representable values uniformly in perceptual space)
- JPEG compression significantly reduces file size at cost of quantization error in high spatial frequencies
  - Human brain is more tolerant of errors in high frequency image components than in low frequency ones
  - Images of the real world are dominated by low-frequency components

H.264 Video Compression

Example video

Go Swallows!

30 second video: 1920 x 1080, @ 30fps

After decode: 8-bits per channel RGB → 24 bits/pixel → 6.2MB/frame
(6.2 MB * 30 sec * 30 fps = 5.2 GB)
Size of data when each frames stored as JPG: 531MB
Actual H.264 video file size: 65.4 MB (80-to-1 compression ratio, 8-to-1 compared to JPG)
Compression/encoding performed in real time on my iPhone

H.264/AVC video compression

- AVC = advanced video coding
- Also called MPEG4 Part 10
- Common format in many modern HD video applications:
  - Blue Ray
  - HD streaming video on internet (Youtube, Vimeo, iTunes store, etc.)
  - HD video recorded by your smart phone
  - European broadcast HDTV (U.S. broadcast HDTV uses MPEG 2)
  - Some satellite TV broadcasts (e.g., DirecTV)
- Benefit: much higher compression ratios than MPEG2 or MPEG4
  - Alternatively, higher quality video for fixed bit rate
- Costs: higher decoding complexity, substantially higher encoding cost
  - Idea: trades off more compute for requiring less bandwidth/storage

Hardware implementations

- Support for H.264 video encode/decode is provided by fixed-function hardware on many modern processors (not just mobile devices)

- Hardware encoding/decoding support existed in modern Intel CPUs since Sandy Bridge (Intel "Quick Sync")

- Modern operating systems expose hardware encode decode support through hardware-accelerated APIs
  - e.g., DirectShow/DirectX (Windows), AVFoundation (iOS)

Video container format versus video codec

- **Video container (MOV, AVI) bundles media assets**

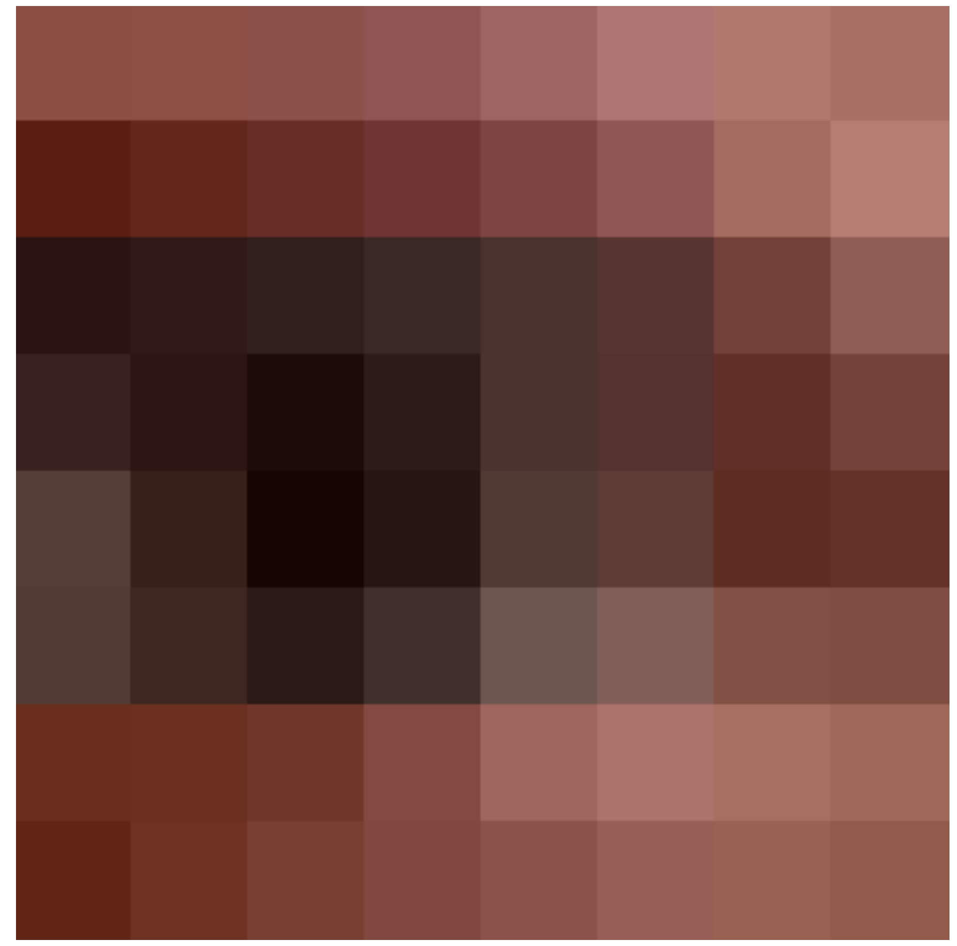
- **Video codec: H.264/AVC (MPEG 4 Part 10)**
  - H.264 standard defines how to represent and decode video
  - H.264 does not define how to encode video (this is left up to implementations)
  - H.264 has many profiles
    - High Profile (HiP): supported by HDV and Blue Ray

Video compression: main ideas

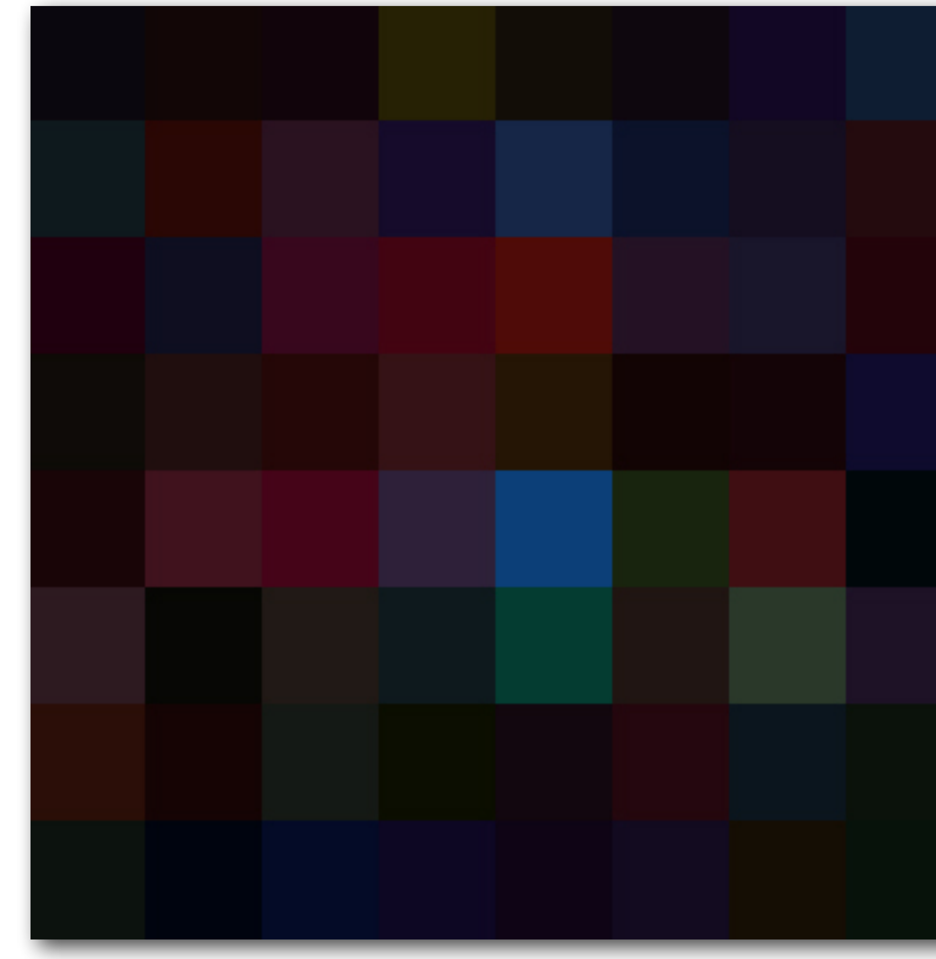
- **Compression is about exploiting redundancy in a signal**
  - **Intra-frame redundancy: value of pixels in neighboring regions of a frame are good <u>predictor</u> of values for other pixels in the frame (spatial redundancy)**
  - **Inter-frame redundancy: pixels from nearby frames in time are a good <u>predictor</u> for the current frame's pixels (temporal redundancy)**

Residual: difference between compressed image and original image

Original pixels

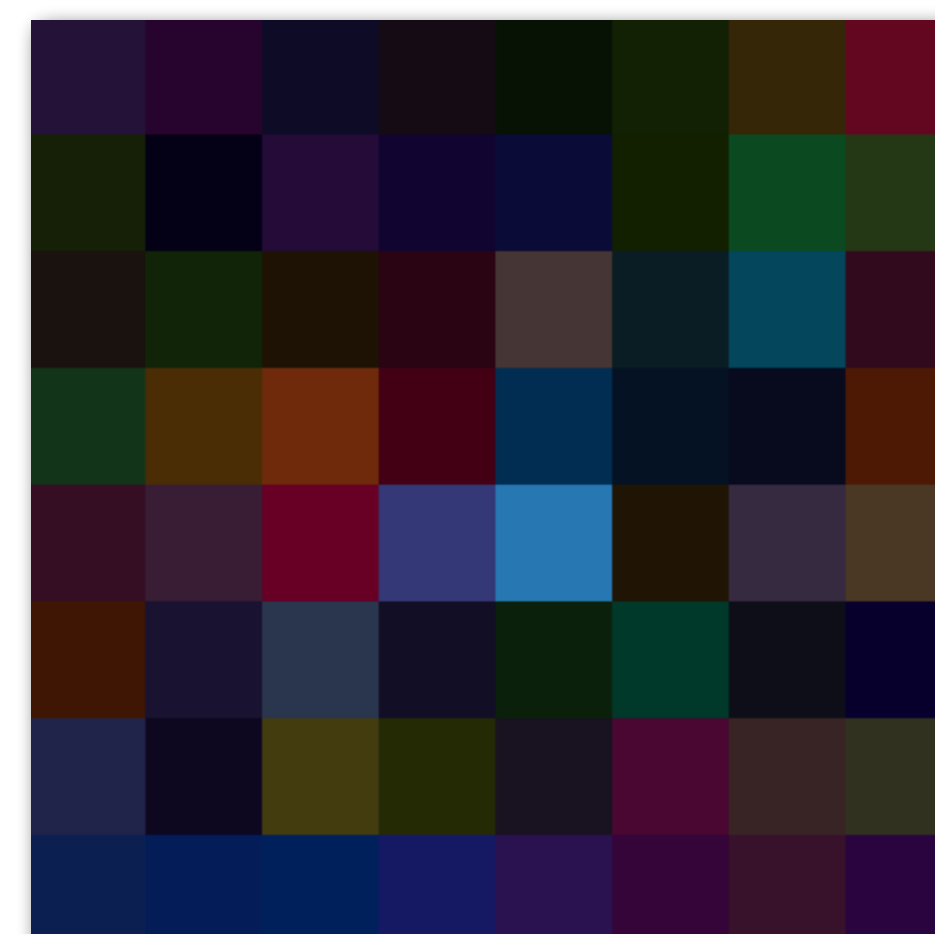


Compressed pixels
(JPEG quality level 6)



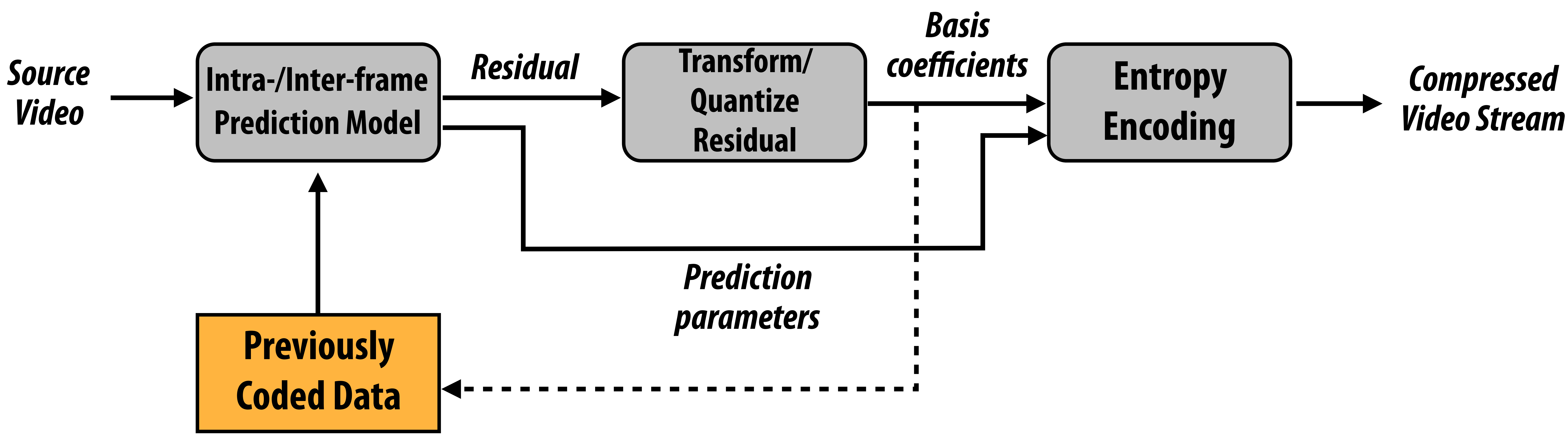
Residual
(amplified for visualization)

Compressed pixels
(JPEG quality level 2)



Residual
(amplified for visualization)

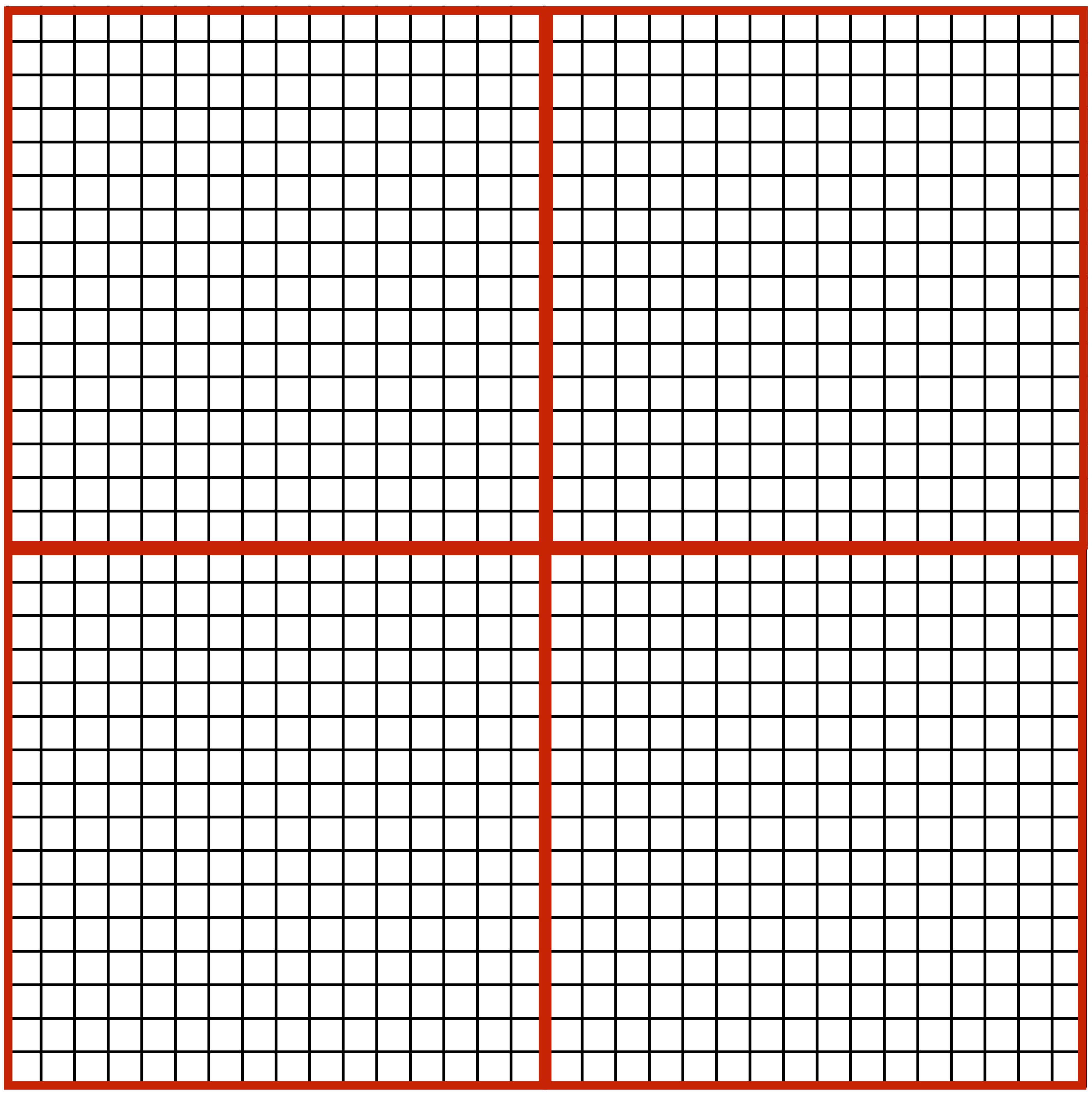
H.264/AVC video compression overview



Residual: difference between predicted pixel values and input video pixel values

Credit: Figure derived from H.264 Advanced Video Compression Standard, I. Richardson, 2010

16 x 16 macroblocks



Video frame is partitioned into 16 x 16 pixel macroblocks

Due to 4:2:0 chroma subsampling, macroblocks correspond to 16 x 16 luma samples and 8 x 8 chroma samples

Macroblocks in an image are organized into slices

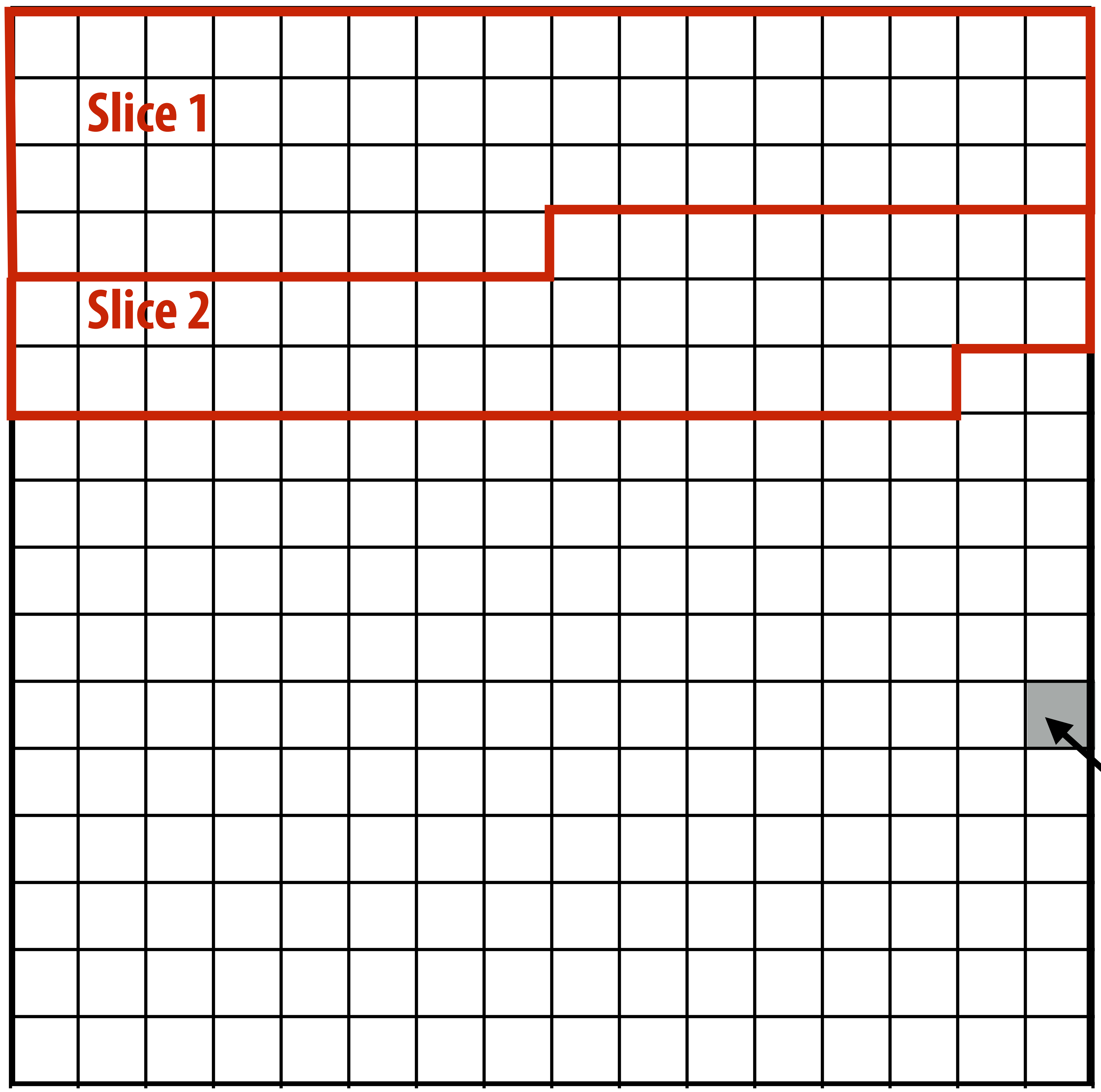


Figure to left shows the macro blocks in a frame

Macroblocks are grouped into slices

Can think of a slice as a sequence of macroblocks in raster scan order *

Slices can be decoded independently **

One 16x16 macroblock

* H.264 also has non-raster-scan order modes (FMO), will not discuss today.

** Final "deblocking" pass is often applied to post-decode pixel data, so technically slices are not fully independent.

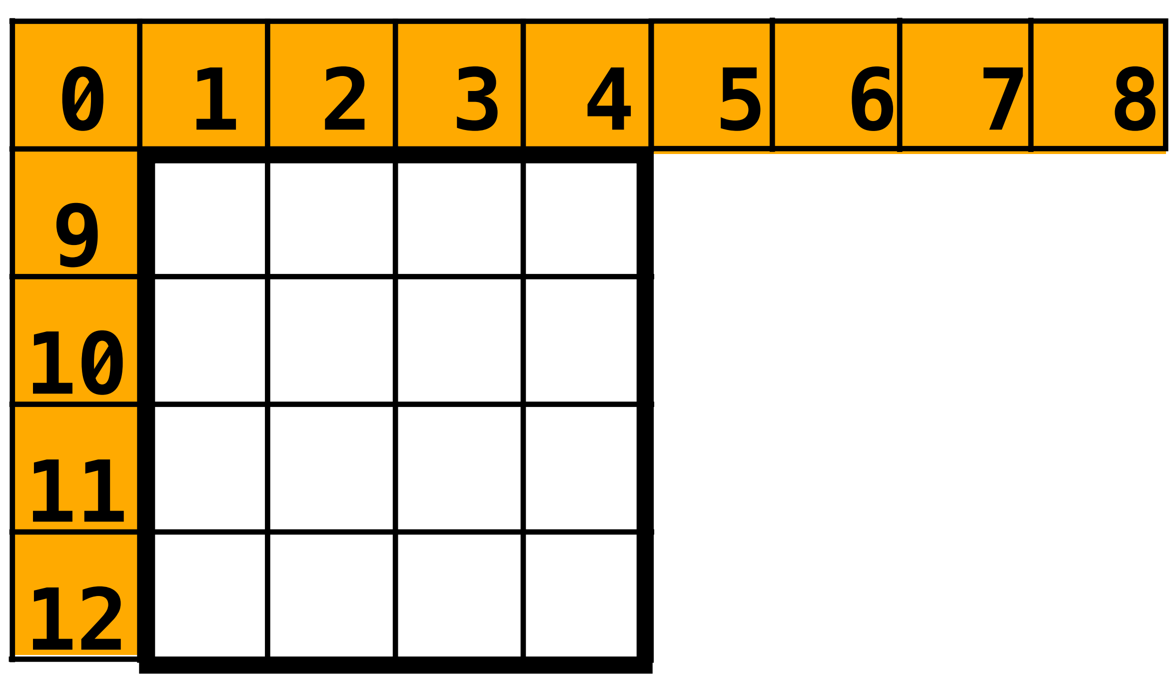
Decoding via prediction + correction

- **During decode, samples in a macroblock are generated by:**
  1. **Making a prediction based on already decoded samples in macroblocks from the same frame (<u>intra-frame prediction</u>) or from other frames (<u>inter-frame prediction</u>)**
  2. **Correcting the prediction with a residual stored in the video stream**

- **Three forms of prediction:**
  - **<u>I-macroblock</u>: macroblock samples predicted from samples in previous macroblocks in the same slice of the current frame**
  - **<u>P-macroblock</u>: macroblock samples can be predicted from samples from one other frame (one prediction per macroblock)**
  - **<u>B-macroblock</u>: macroblock samples can be predicted by a weighted combination of multiple predictions from samples from other frames**

Intra-frame prediction (I-macroblock)

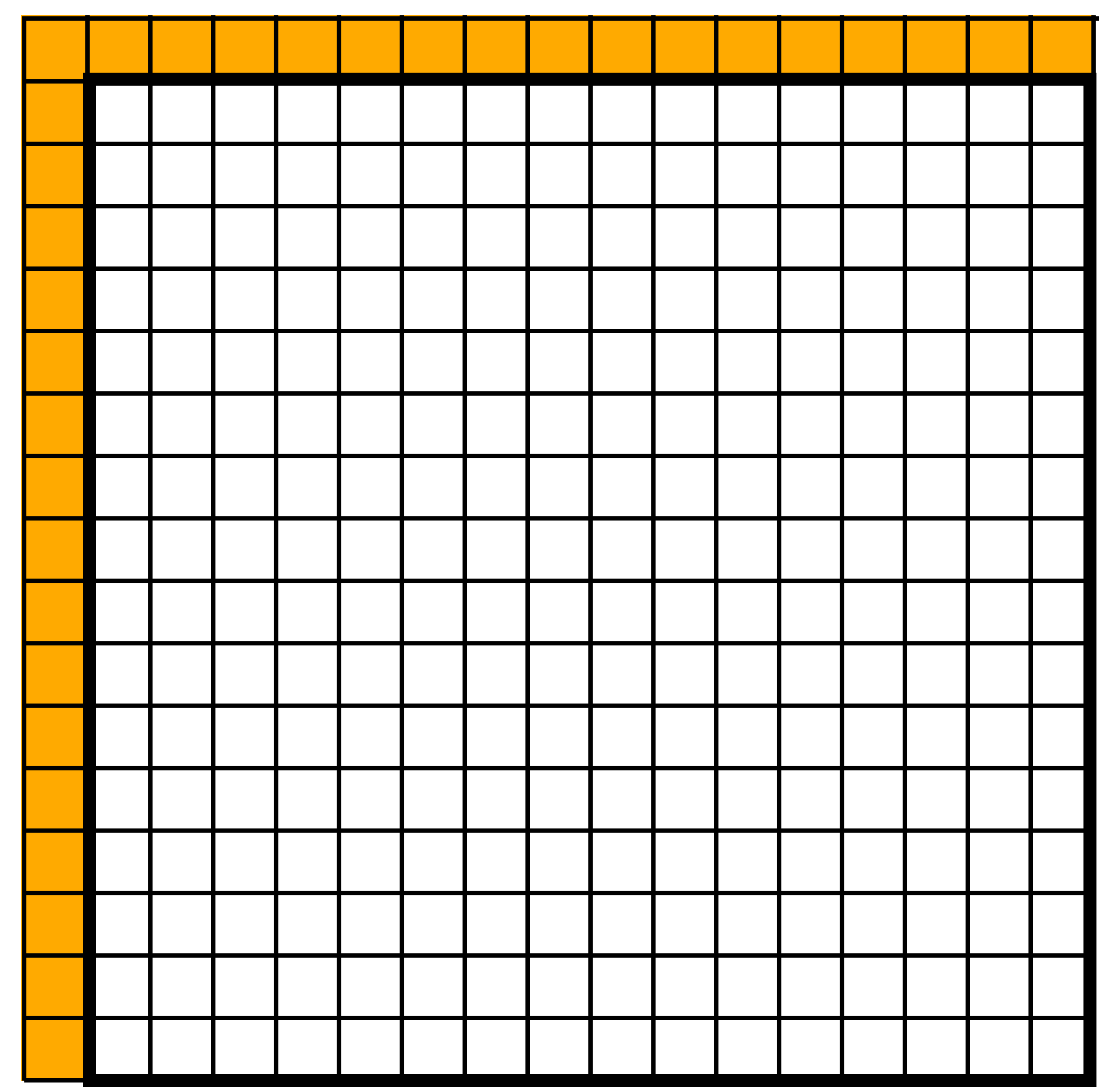
- Prediction of sample values is performed in spatial domain, not transform domain
  - Predicting pixel values, not basis coefficients
- Modes for predicting the 16x16 luma (Y) values: *
  - Intra_4x4 mode: predict 4x4 block of samples from adjacent row/col of pixels
  - Intra_16x16 mode: predict entire 16x16 block of pixels from adjacent row/col
  - I_PCM: actual sample values provided



Intra_4X4

Yellow pixels: already reconstructed (values known)

White pixels: 4x4 block to be reconstructed

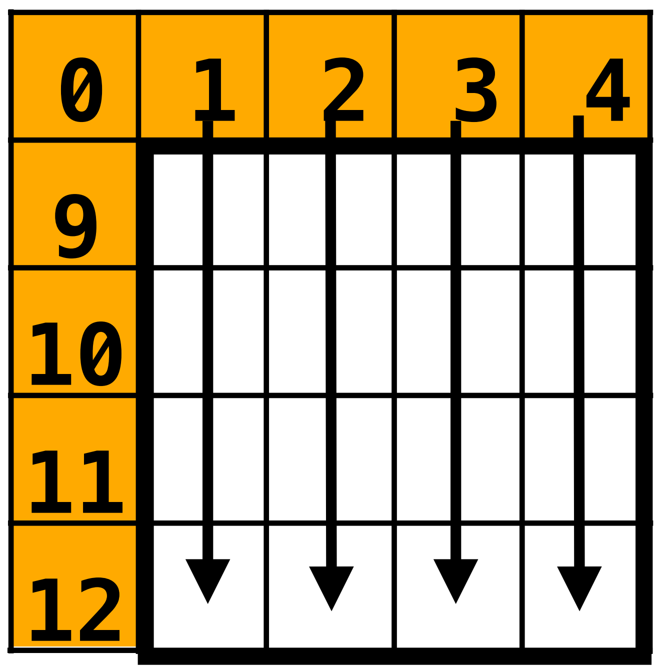


Intra_16x16

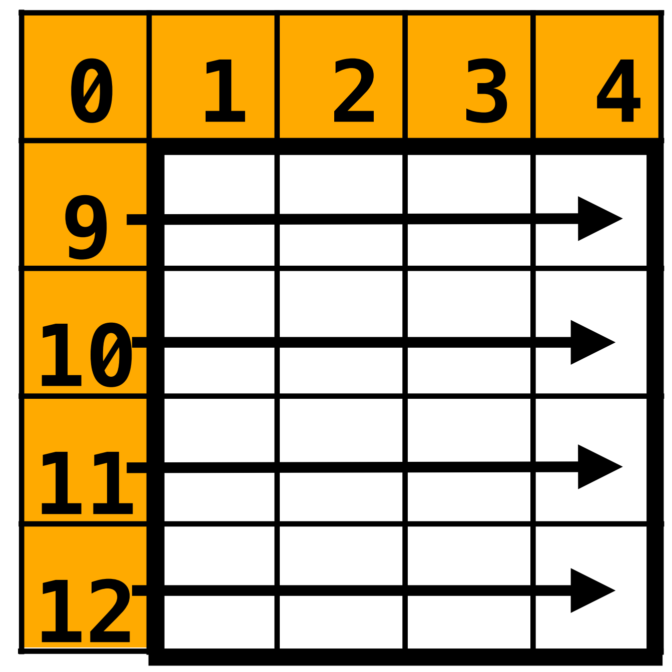
* An additional 8x8 mode exists in the H.264 High Profile

Intra_4x4 prediction modes

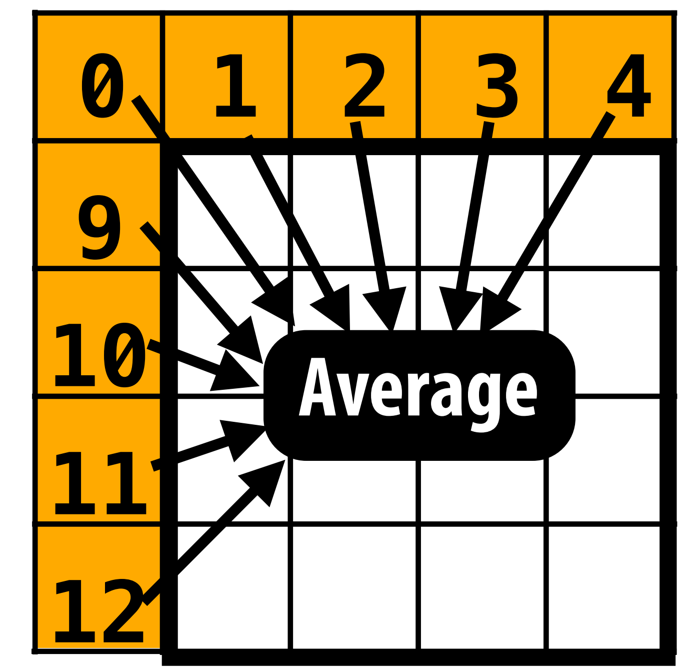
- Nine prediction modes (6 shown below)
  - Other modes: horiz-down, vertical-left, horiz-up



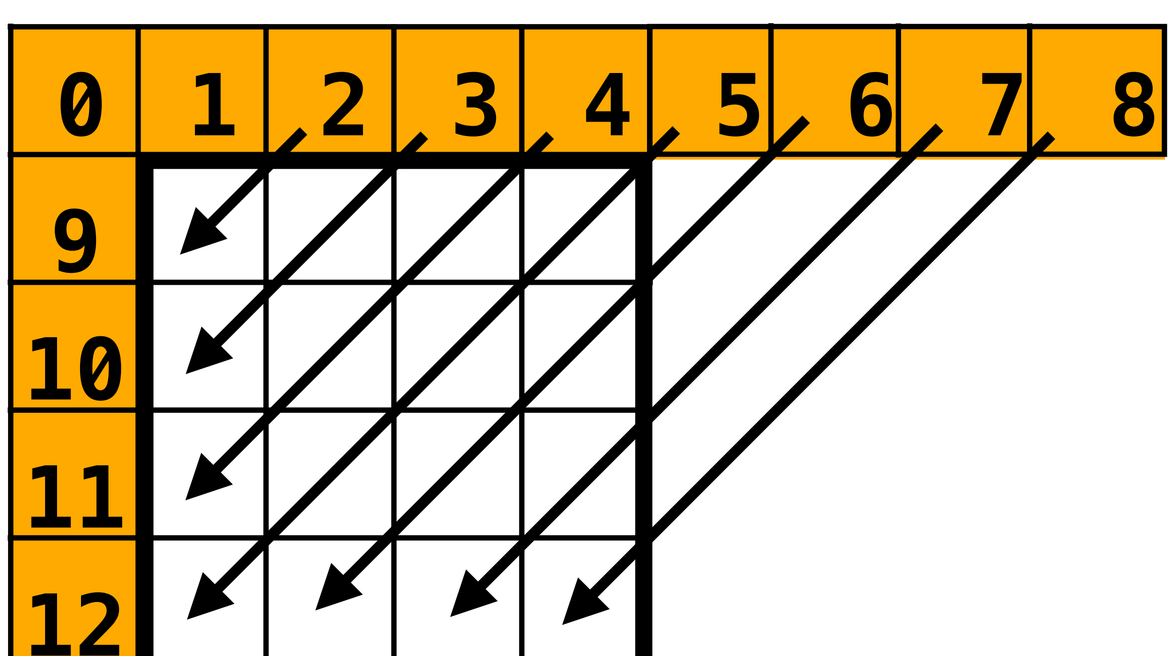
Mode 0: vertical
(4x4 block is copy of above row of pixels)



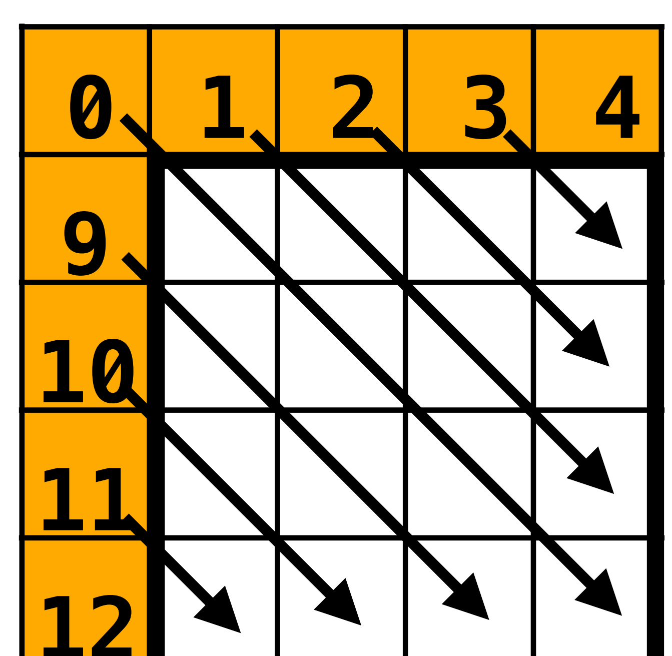
Mode 1: horizontal
(4x4 block is copy of left col of pixels)



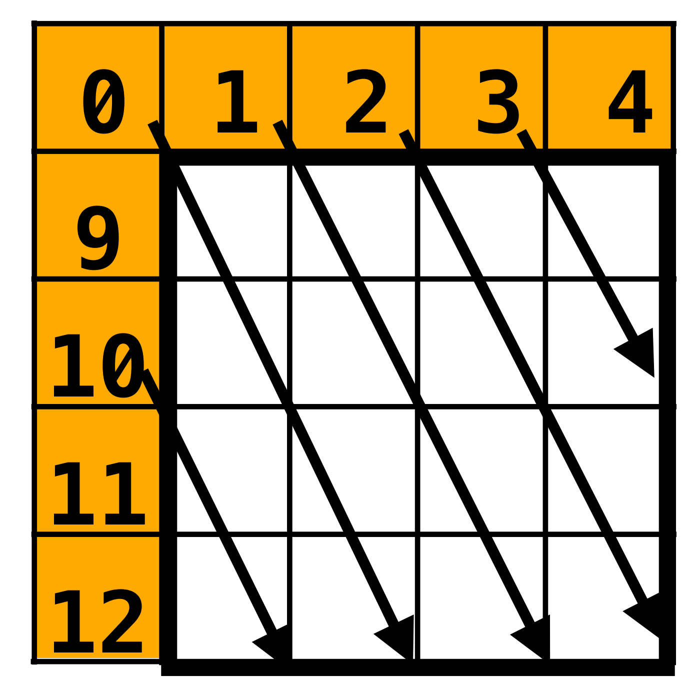
Mode 2: DC
(4x4 block is average of above row and left col of pixels)



Mode 3: diagonal down-left (45°)



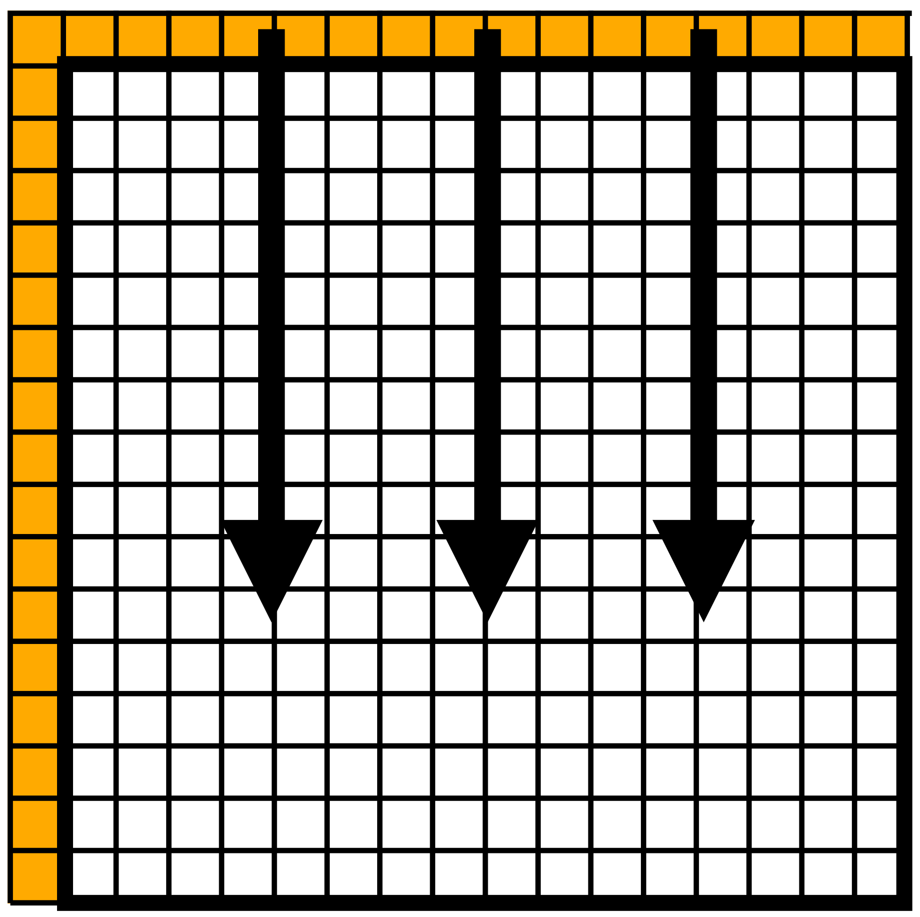
Mode 4: diagonal down-right (45°)



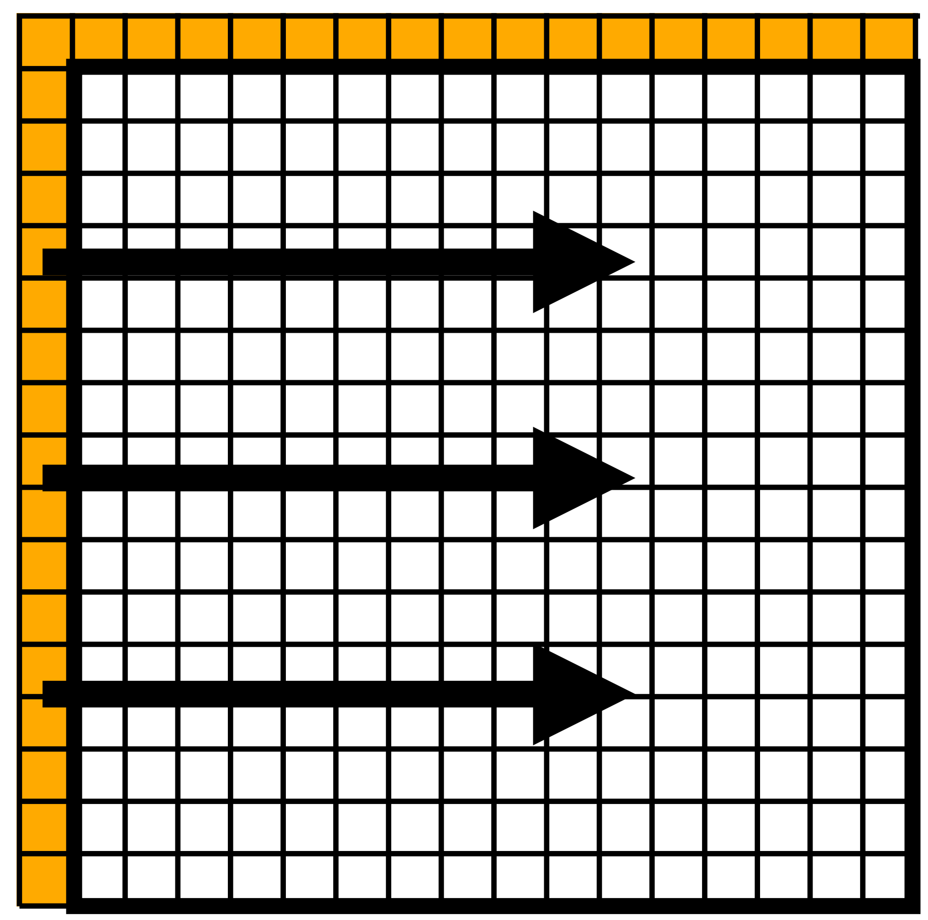
Mode 5: vertical-right (26.6°)

Intra_16x16 prediction modes

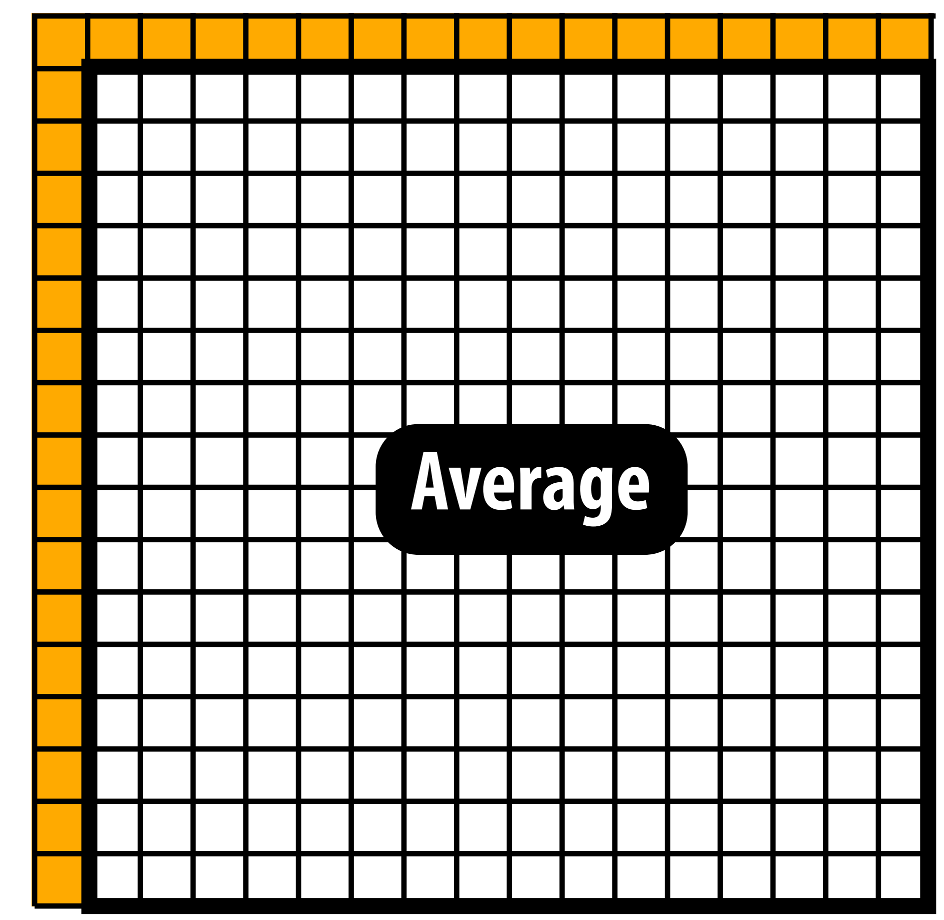
- 4 prediction modes: vertical, horizontal, DC, plane



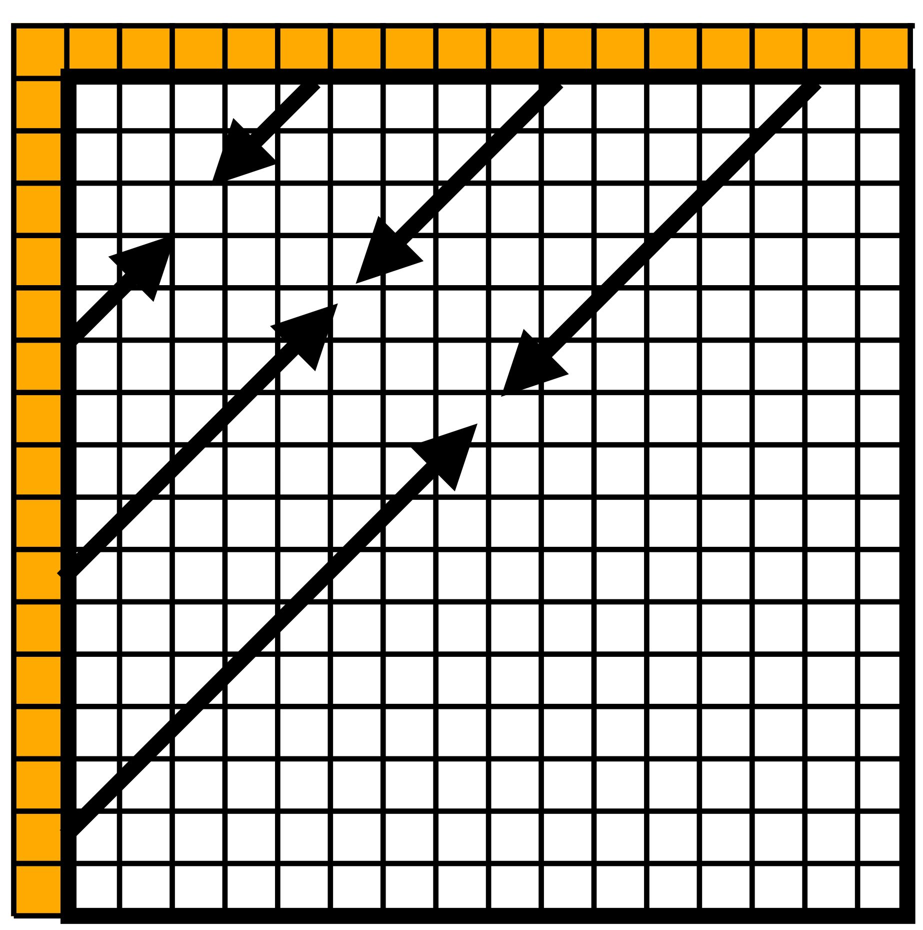
Mode 0: vertical



Mode 1: horizontal



Mode 2: DC



Mode 4: plane

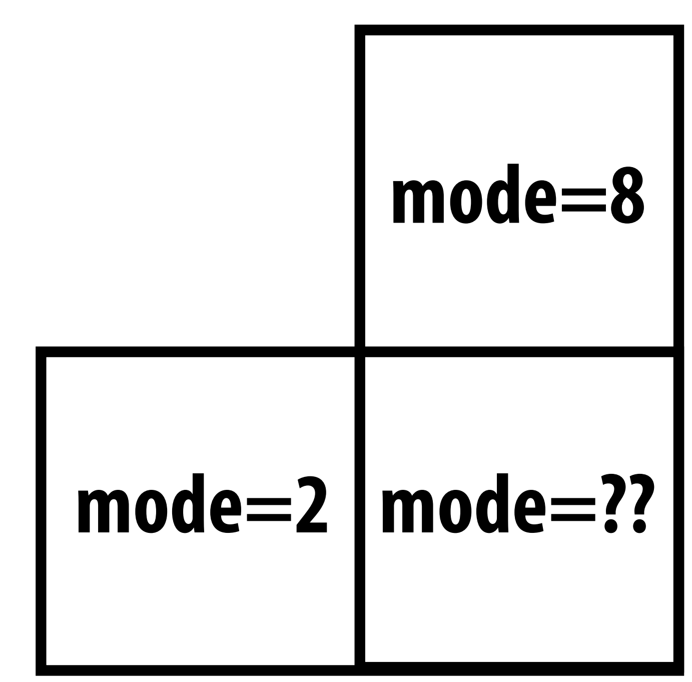
P[i,j] = Ai * Bj + C

A derived from top row, B derived from left col, C from both

Further details

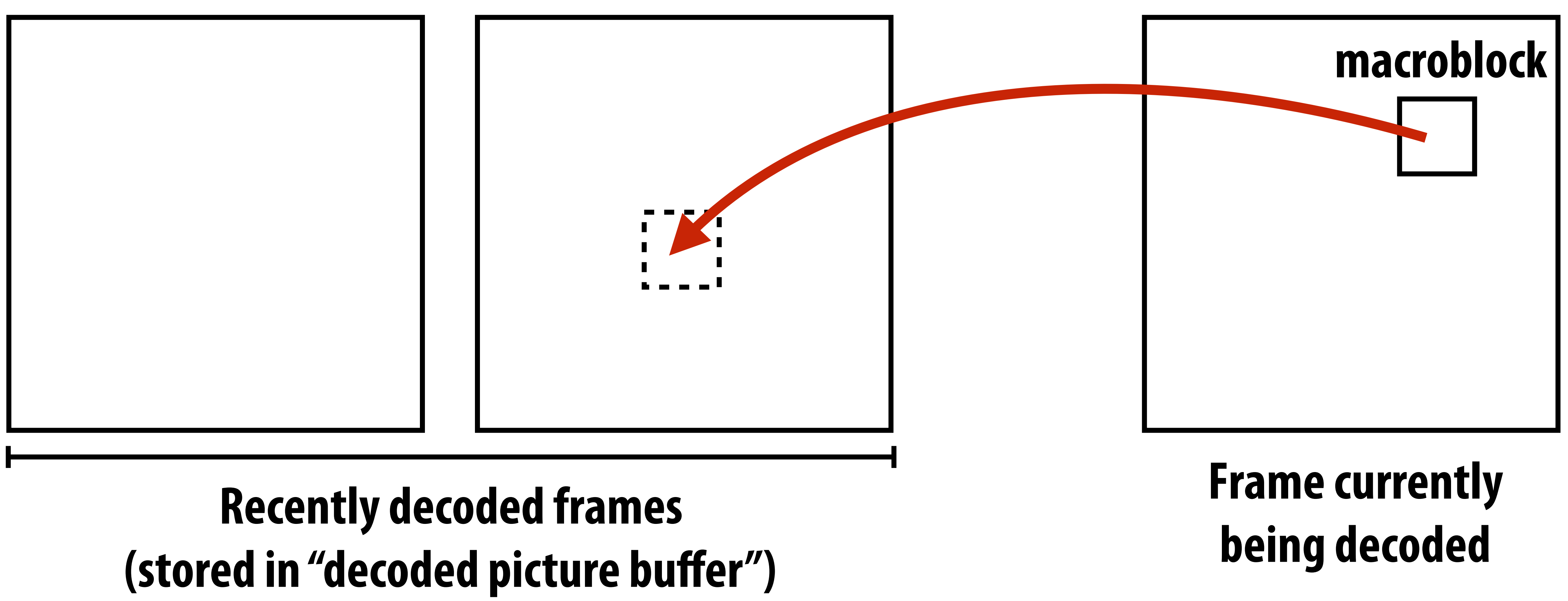
- Intra-prediction of chroma (8x8 block) is performed using four modes similar to those of intra_16x16 (except reordered as: DC, vertical, horizontal, plane)

- Intra-prediction scheme for each 4x4 block within macroblock encoded as follows:
  - One bit per 4x4 block:
    - if 1, use <u>most probable</u> mode
      - Most probable = lower of modes used for 4x4 block to left or above current block
    - if 0, use additional 3-bit value `rem_intra4x4_pred_mode` to encode one of nine modes
      - if `rem_intra4x4_pred_mode` is smaller than most probable mode, use mode given by `rem_intra4x4_pred_mode`
      - else, mode is `rem_intra4x4_pred_mode+1`



Inter-frame prediction (P-macroblock)

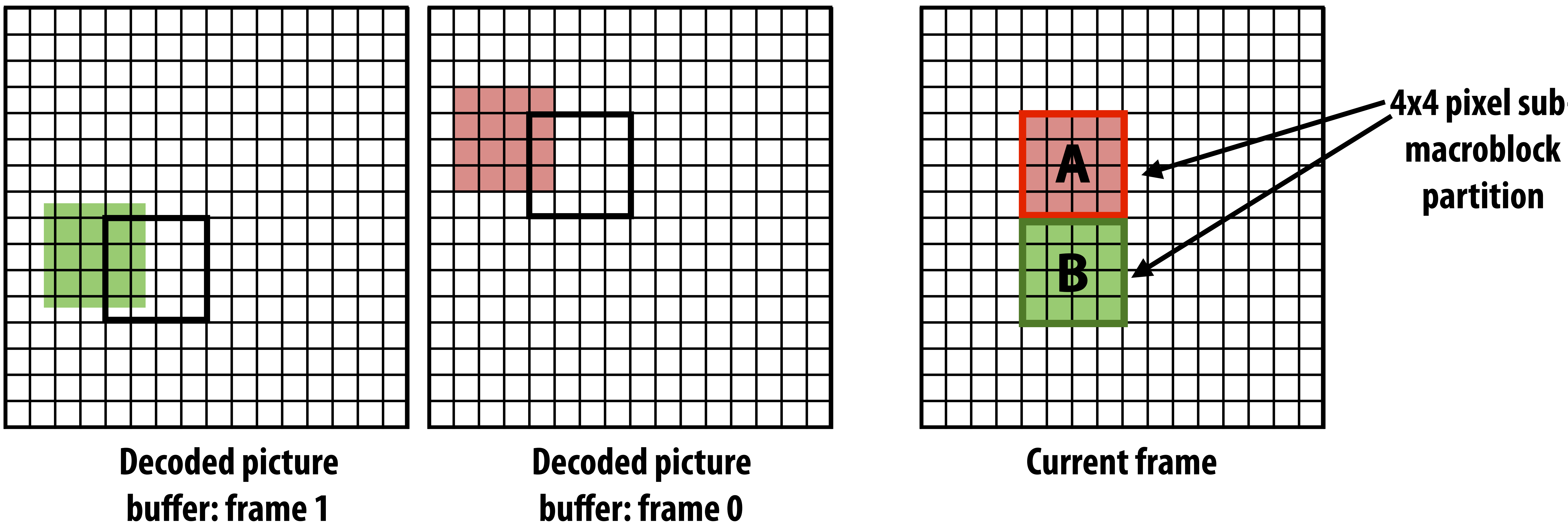
- Predict sample values using values from a block of a previously decoded frame *
- Basic idea: current frame formed by translation of pixels from temporally nearby frames (e.g., object moved slightly on screen between frames)
  - "Motion compensation": use of spatial displacement to make prediction about pixel values



* Note: "previously decoded" does not imply source frame must come before current frame in the video sequence. (H.264 supports decoding out of order.)

P-macroblock prediction

- Prediction can be performed at macroblock or sub-macroblock granularity
  - Macroblock can be divided into 16x16, 8x16, 16x8, 8x8 "partitions"
  - 8x8 partitions can be further subdivided into 4x8, 8x4, 4x4 sub-macroblock partitions
- Each partition predicted by sample values defined by:
(reference frame id, motion vector)



Block A: predicted from (frame 0, motion-vector = [-3, -1])

Block B: predicted from (frame 1, motion-vector = [-2.5, -0.5])

Note: non-integer motion vector

Motion vector visualization

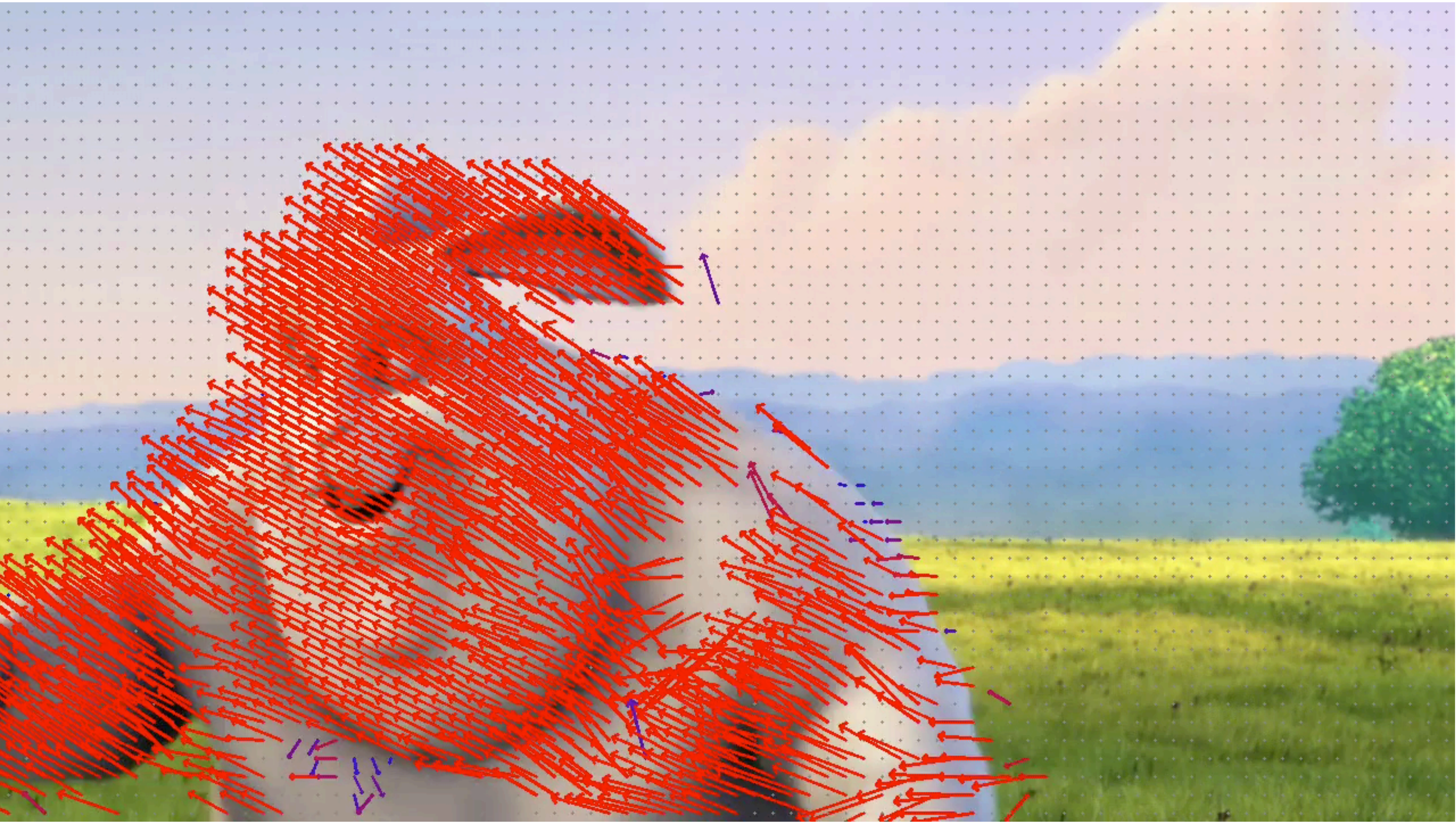
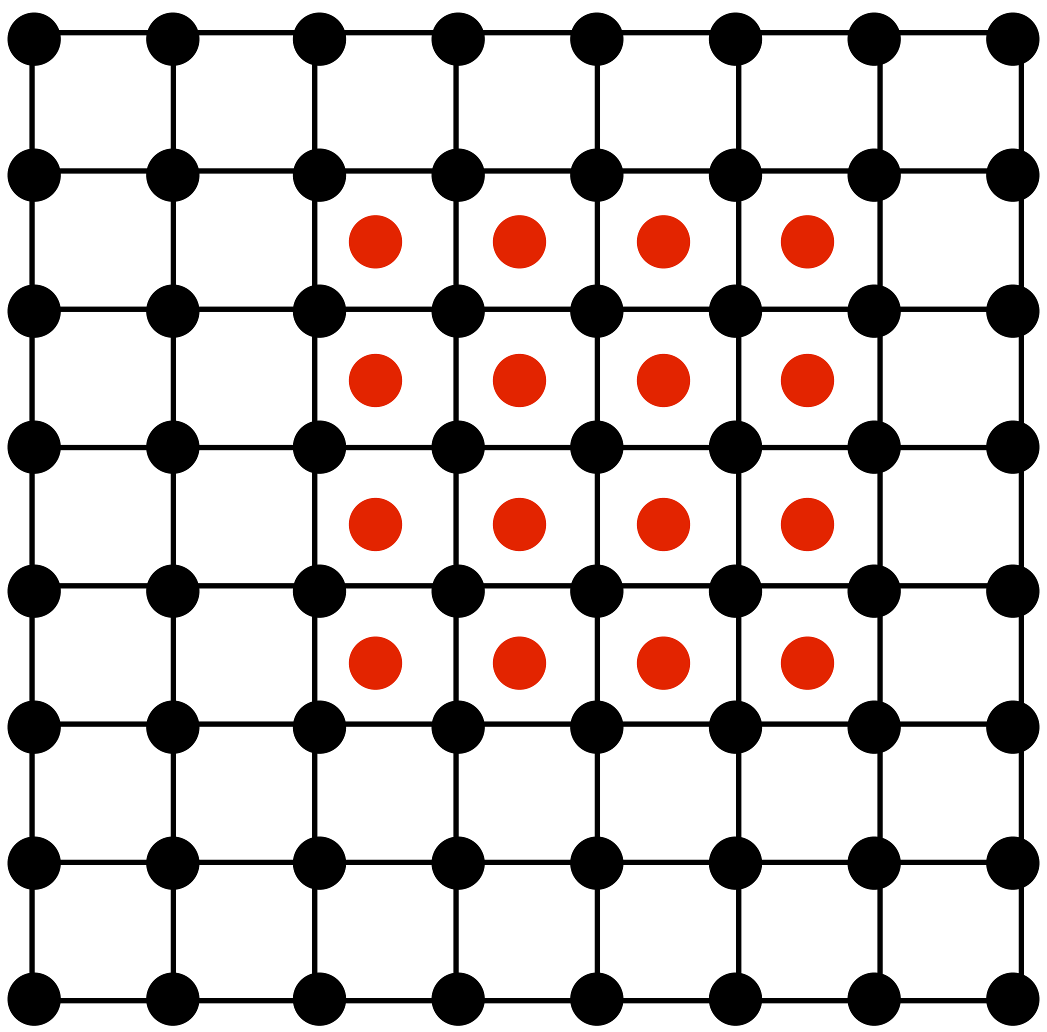


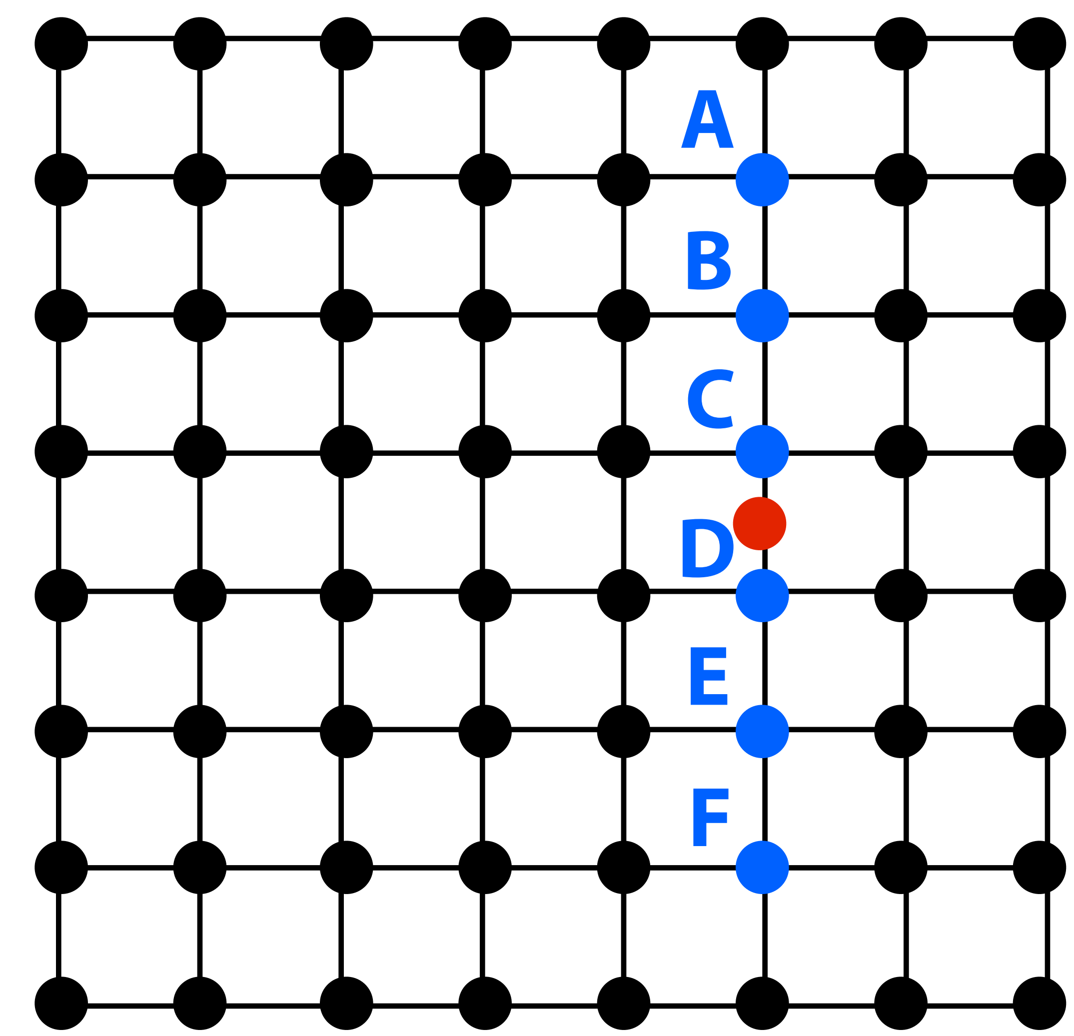
Image credit: Keyi Zhang

Non-integer motion vectors require resampling



Example: motion vector with 1/2 pixel values.

Must resample reference block at positions given by red dots.



Interpolation to 1/2 pixel sample points via 6-tap filter:

half_integer_value = clamp((A - 5B + 20C + 20D - 5E + F) / 32)

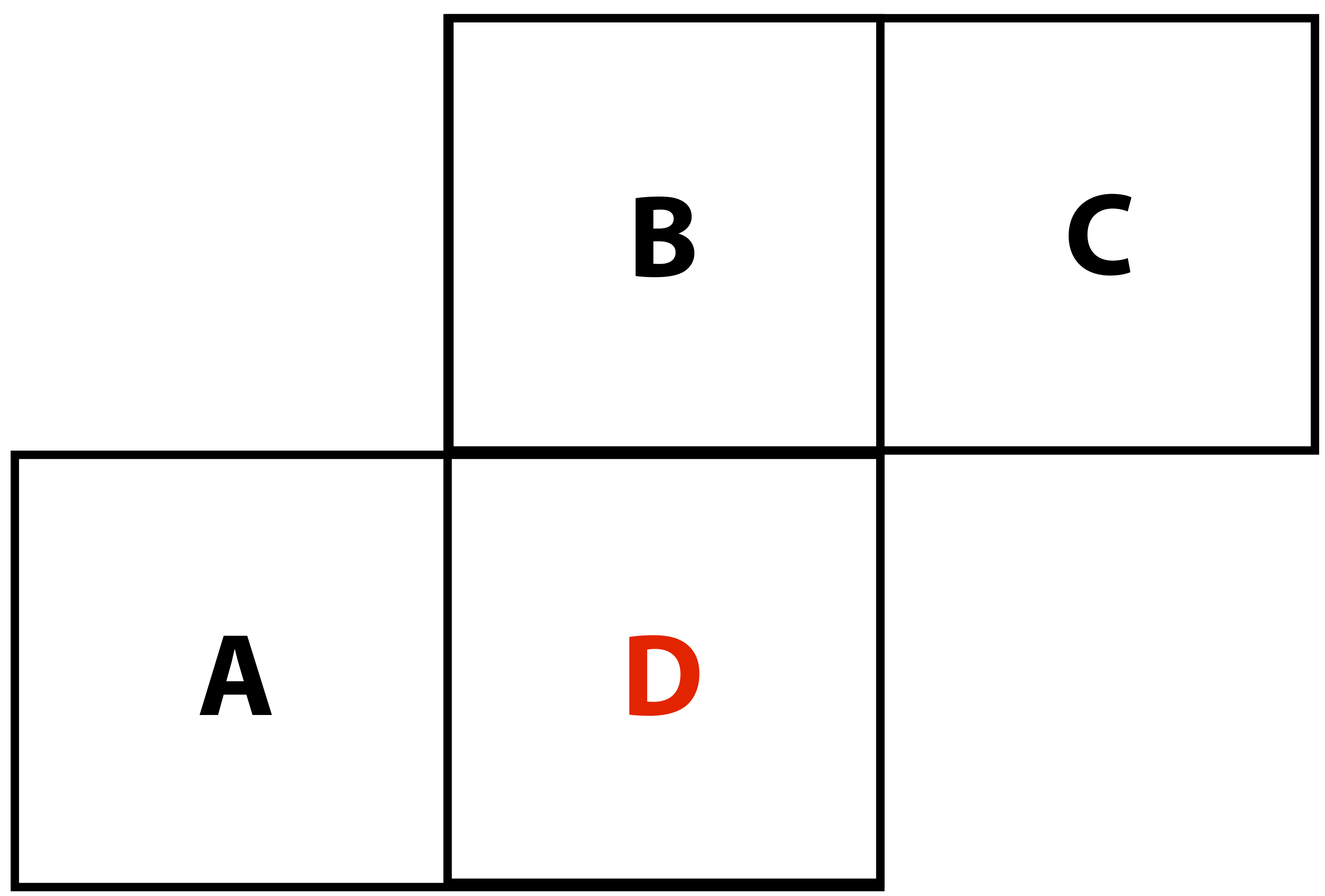
H.264 supports both 1/2 pixel and 1/4 pixel resolution motion vectors

1/4 resolution resampling performed by bilinear interpolation of 1/2 pixel samples

1/8 resolution (chroma only) by bilinear interpolation of 1/4 pixel samples

Motion vector prediction

- Problem: per-partition motion vectors require significant amount of storage
- Solution: predict motion vectors from neighboring partitions and encode residual in compressed video stream
  - Example below: predict D's motion vector as average of motion vectors of A, B, C
  - Prediction logic becomes more complex when when partitions of neighboring blocks are of different size

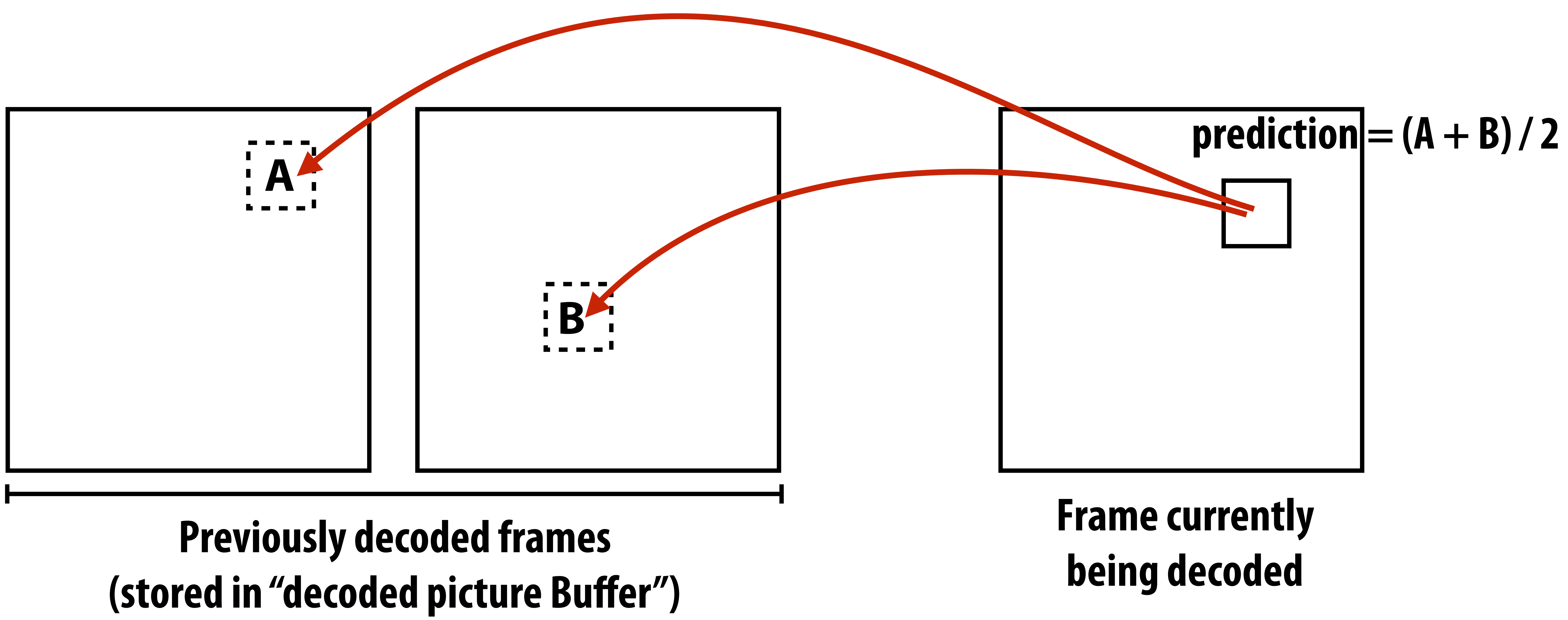


Question: what partition size is best?

- Smaller partitions likely yield more accurate prediction
  - Fewer bits needed for residuals
- Smaller partitions require more bits to store partition information (diminish benefits of prediction)
  - Reference picture id
  - Motion vectors (note: motion vectors are more coherent with finer sampling, so they likely compress well)

Inter-frame prediction (B-macroblock)

- **Each partition predicted by up to two source blocks**
  - Prediction is the average of the two reference blocks
  - Each B-macroblock partition stores two frame references and two motion vectors (recall P-macroblock partitions only stored one)



Additional prediction details

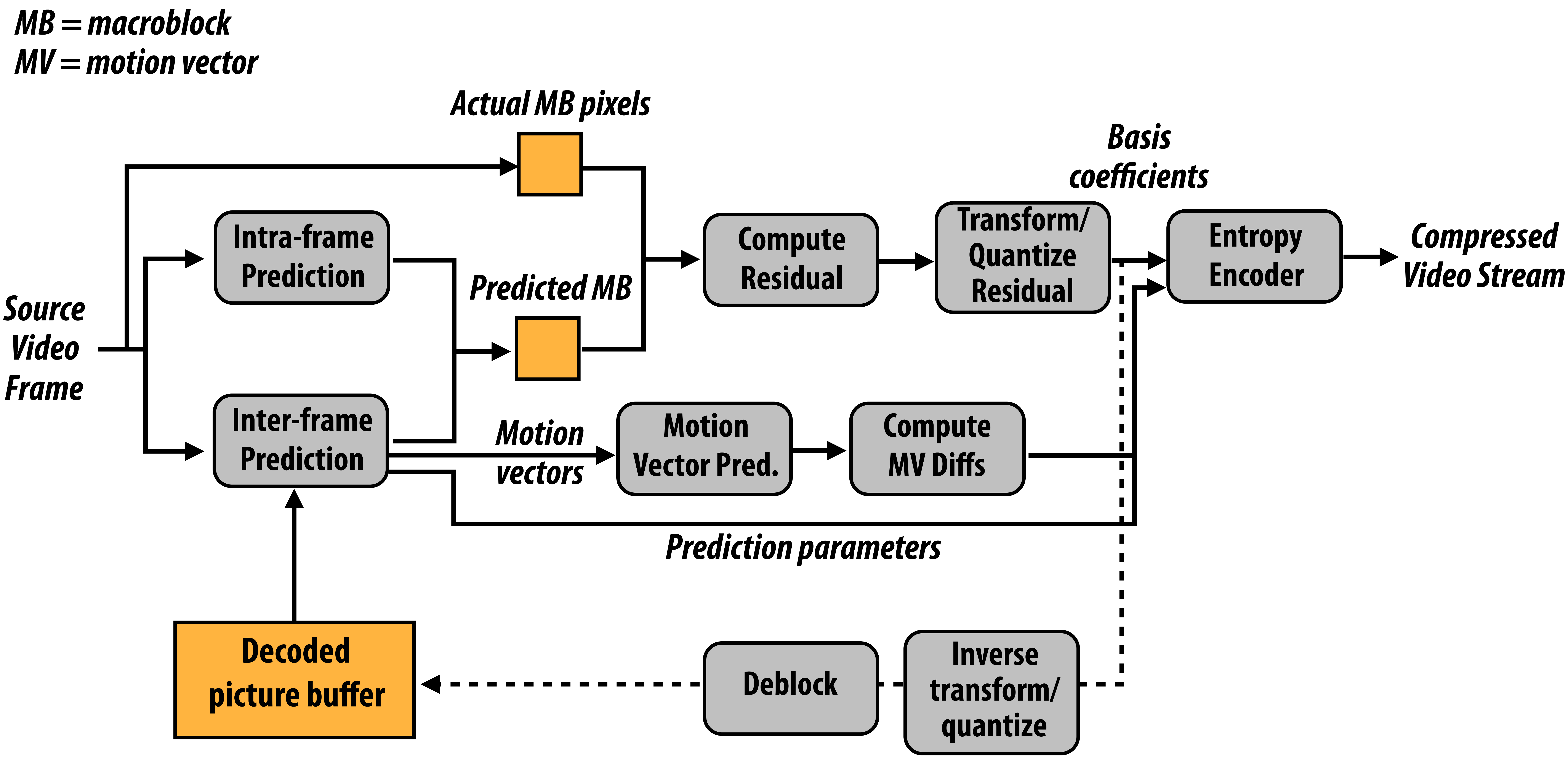
- **Optional weighting to prediction:**
  - Per-slice explicit weighting (reference samples multiplied by weight)
  - Per-B-slice implicit weights (reference samples weights by temporal distance of reference frame from current frame in video)
    - Idea: weight samples from reference frames nearby in time more
- **Deblocking**
  - Blocking artifacts may result as a result of macroblock granularity encoding
  - After macroblock decoding is complete, optionally perform smoothing filter across block edges.

Putting it all together: encoding an inter-predicted macroblock

- Inputs:
  - Current state of decoded picture buffer (state of the decoder)
  - 16x16 block of input video to encode
- General steps: (need not be performed in this order)
  - Resample images in decoded picture buffer to obtain 1/2, and 1/4, 1/8 pixel resampling
  - Choose prediction type (P-type or B-type)
  - Choose reference pictures for prediction
  - Choose motion vectors for each partition (or sub-partition) of macroblock
  - Predict motion vectors and compute motion vector difference
  - Encode choice of prediction type, reference pictures, and motion vector differences
  - Encode residual for macroblock prediction
  - Store reconstructed macroblock (post deblocking) in decoded picture buffer to use as reference picture for future macroblocks

Coupled decisions

H.264/AVC video encoding



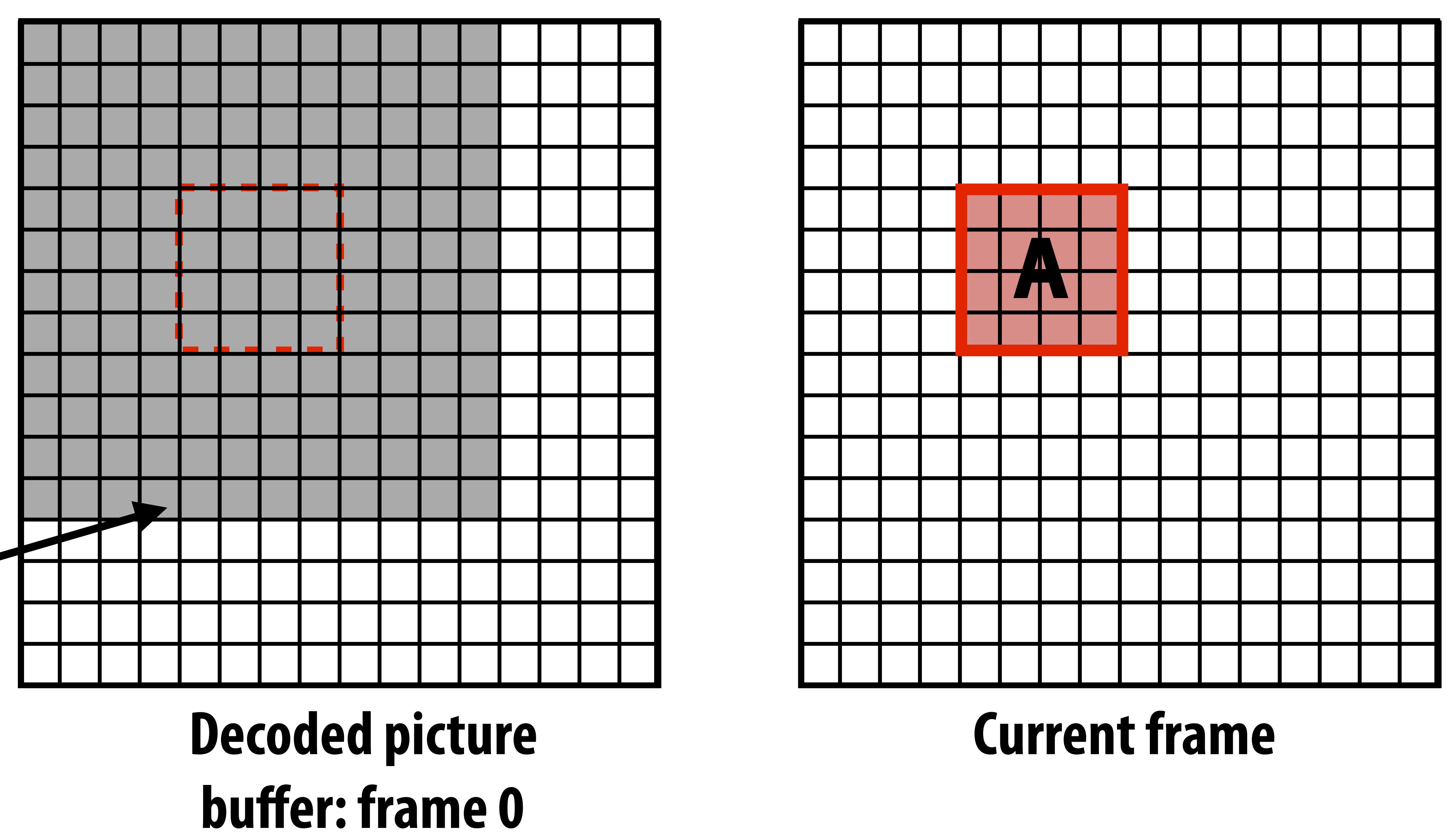
Credit: Figure derived from <u>H.264 Advanced Video Compression Standard</u>, I. Richardson, 2010

Motion estimation

- Encoder must <u>find</u> reference block that predicts current frame's pixels well.
  - Can search over multiple pictures in decoded picture buffer + motion vectors can be non-integer (huge search space)
  - Must also choose block size (macroblock partition size)
  - And whether to predict using combination of two blocks
  - Literature is full of heuristics to accelerate this process
    - Remember, must execute motion estimation in real-time for HD video (1920x1080), on a low-power smartphone

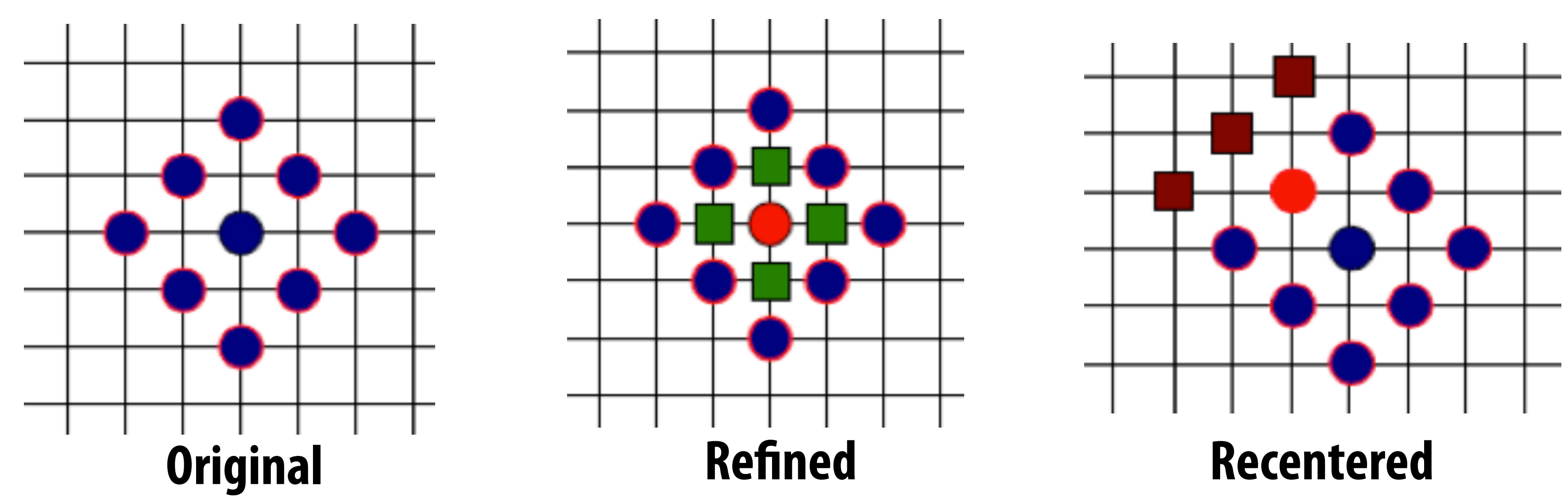
Limit search window:

gray area: search region



Motion estimation optimizations

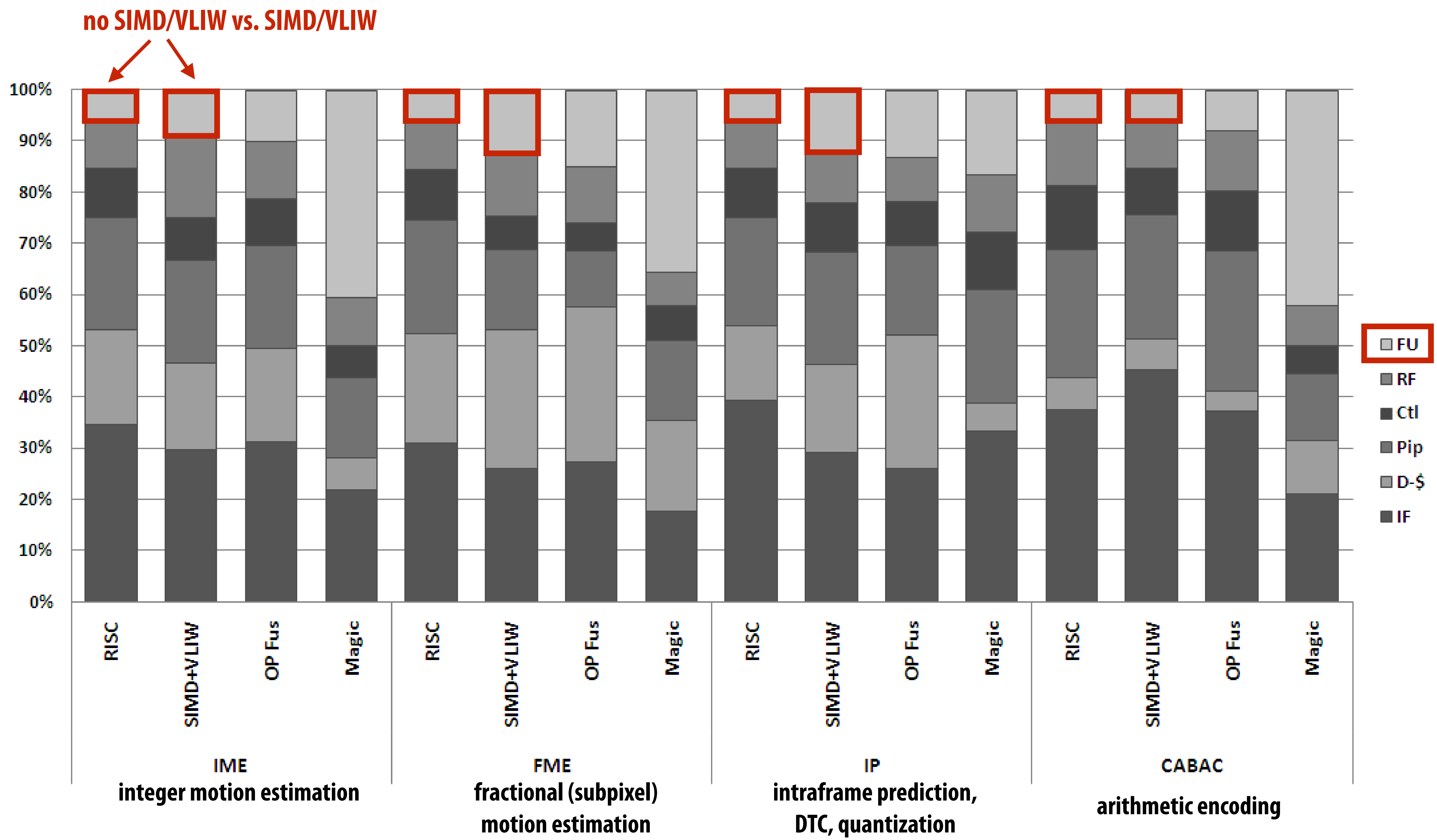
- Coarser search:
  - Limit search window to small region
  - First compute block differences at coarse scale (save partial sums from previous searches)
- Smarter search:
  - Guess motion vectors similar to motion vectors used for neighboring blocks
  - Diamond search: start by test large diamond pattern centered around block
    - If best match is interior, refine to finer scale
    - Else, recenter around best match



- Early termination: don't find optimal reference patch, just find one that's "good enough": e.g., compressed representation is lower than threshold
  - Test zero-motion vector first (optimize for non-moving background)
- Optimizations for subpixel motion vectors:
  - Refinement: find best reference block given only pixel offsets, then try 1/2, 1/4-subpixel offsets around this match

Fraction of energy consumed by different parts of instruction pipeline (H.264 video encoding)

[Hameed et al. ISCA 2010]



56% of total time	36% of total time	7% of total time	1% of total time (of baseline CPU imp)

FU = functional units
RF = register fetch
Ctrl = misc pipeline control
Pip = pipeline registers (interstage)
D-$ = data cache
IF = instruction fetch + instruction cache

H.265 (HVEC)

- Standard ratified in 2013
- Goal: ~2X better compression than H.264
- Main ideas:
  - Macroblock sizes up to 64x64
  - Prediction block size and residual block sizes can be different
  - 35 intra-frame prediction modes (recall H.264 had 9)
  - …

Summary

- JPG image compression and H.264 video compression are "lossy" compression techniques that discard information is that less likely to be noticed by the human eye
  - Key principle: "Lossy, but still looks good enough to humans!"

- But moving forward most videos in the world will be analyzed by computers, not viewed by humans
  - What is appropriate form of compression for a particular set of computer vision tasks?
  - New principle: "Lossy, but image analysis tasks still work!"
  - Can we "learn" domain-specific compressors for particular scenes, types of tasks, etc?