Lecture 10:

Video Compression (Traditional and Learned)

Visual Computing Systems

Stanford CS348K, Spring 2022

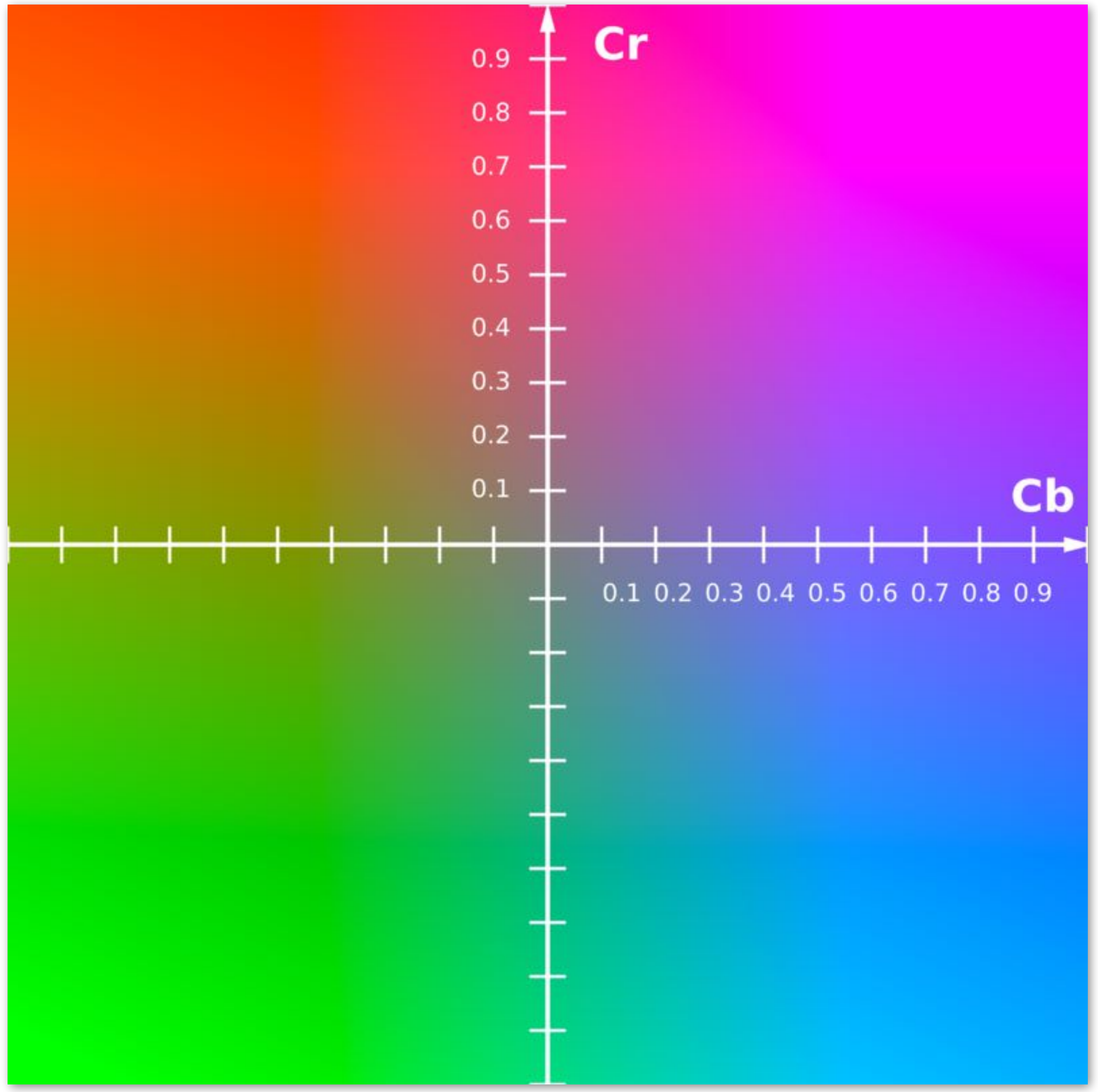
Image compression review/fundamentals

Y'CbCr color space

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

Cb = blue-yellow deviation from gray

Cr = red-cyan deviation from gray



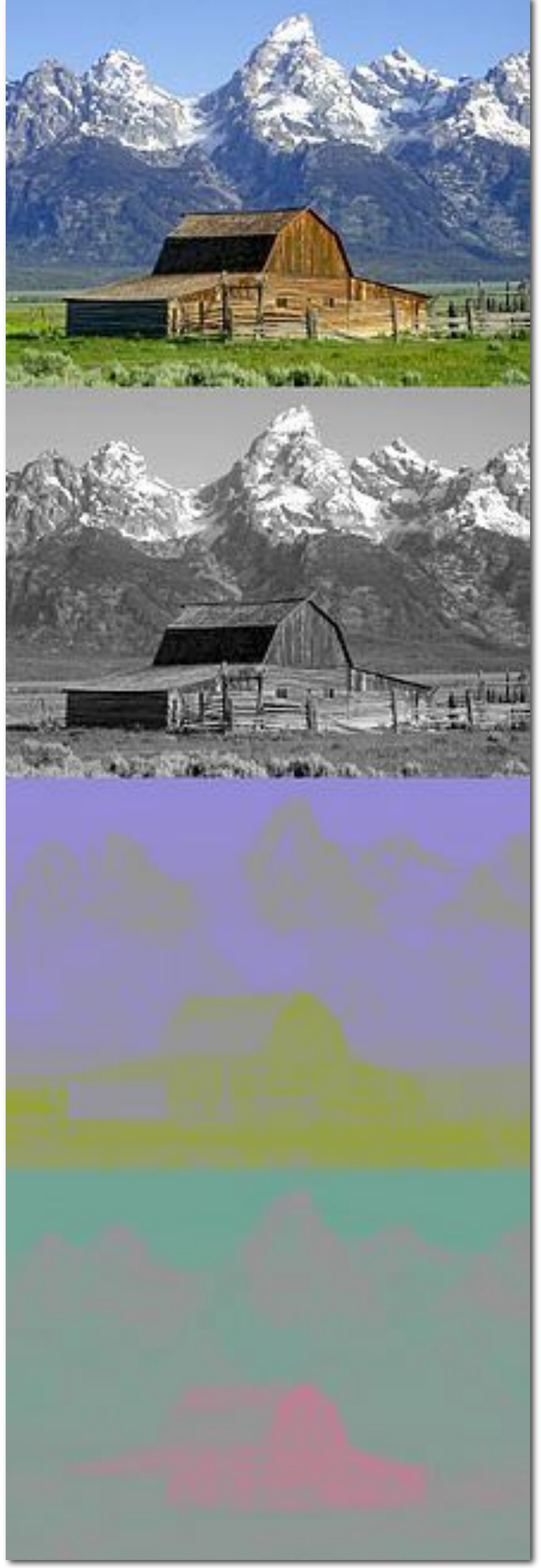
Non-linear RGB
(primed notation indicates perceptual (non-linear) space)

Conversion from R'G'B' to Y'CbCr:

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

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

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



Y'

Cb

Cr

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

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

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

Real-world 4:2:0 examples:
most JPG images and H.264 video

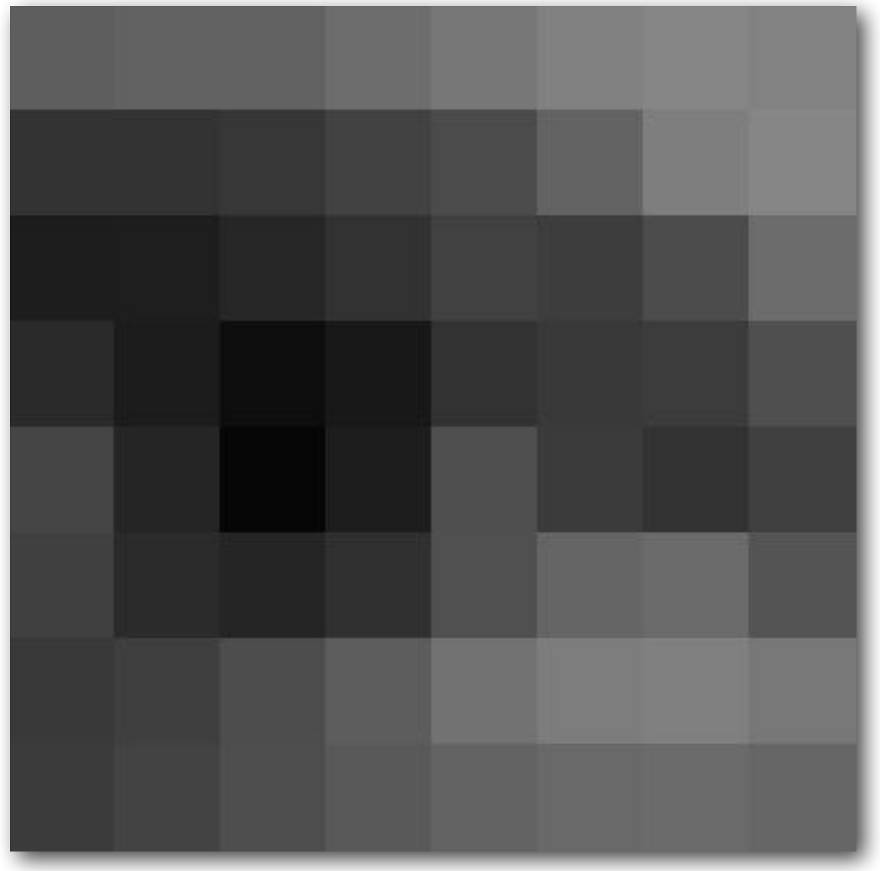
Image transform coding using the discrete cosign transform (DCT)

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

64 basis coefficients

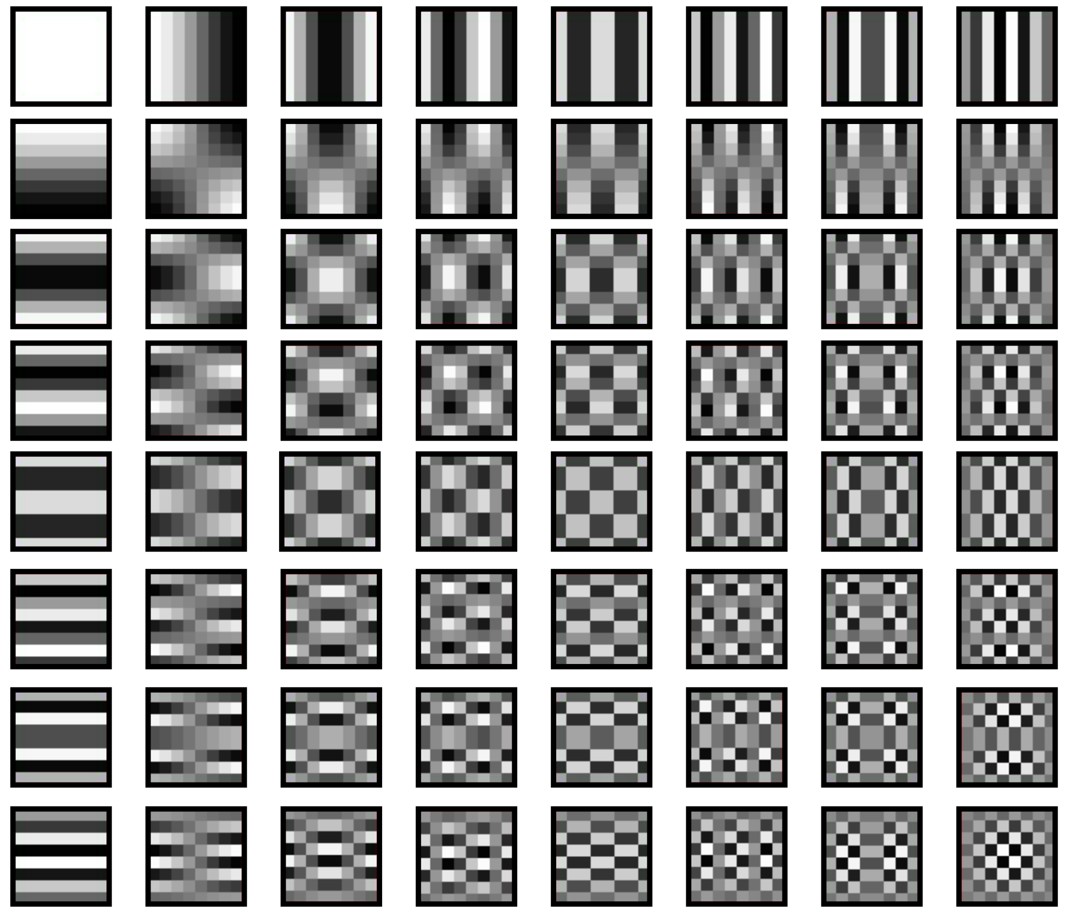
64 cosine basis vectors (each vector is 8x8 image)

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



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

[0,0]



[7,7]

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

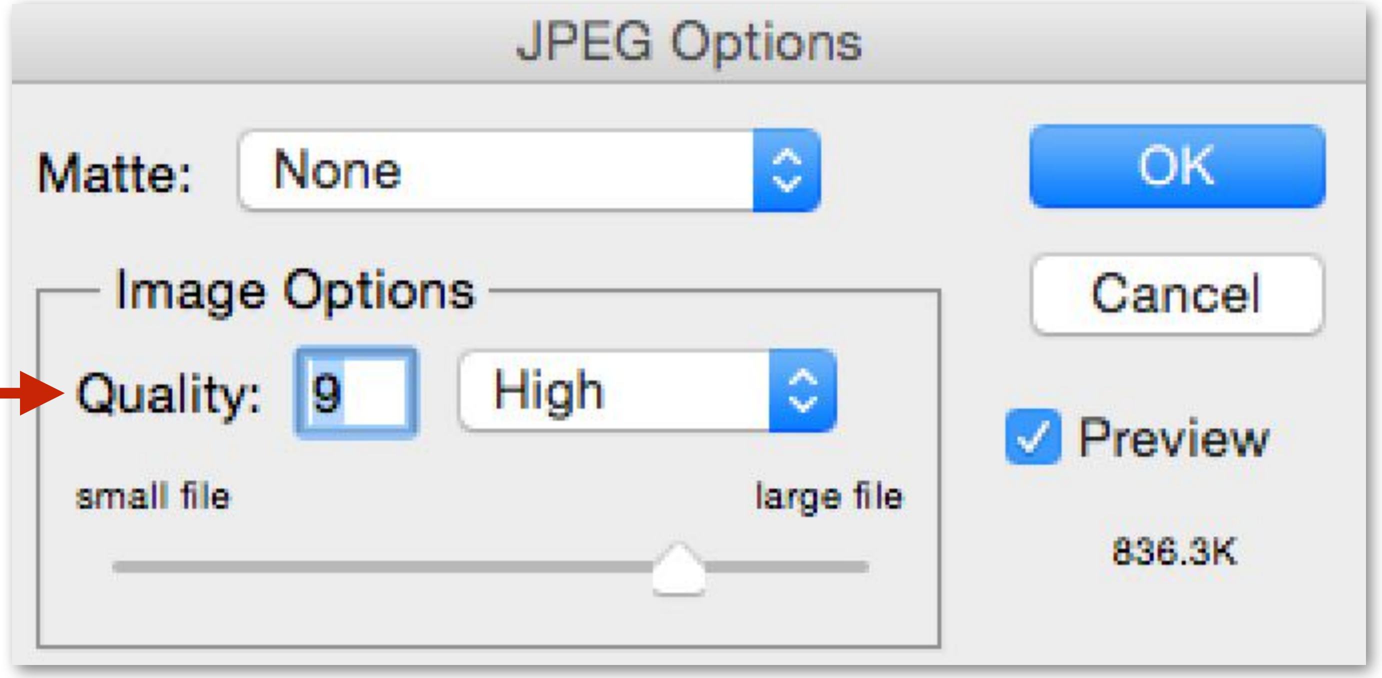
Quantization

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

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)

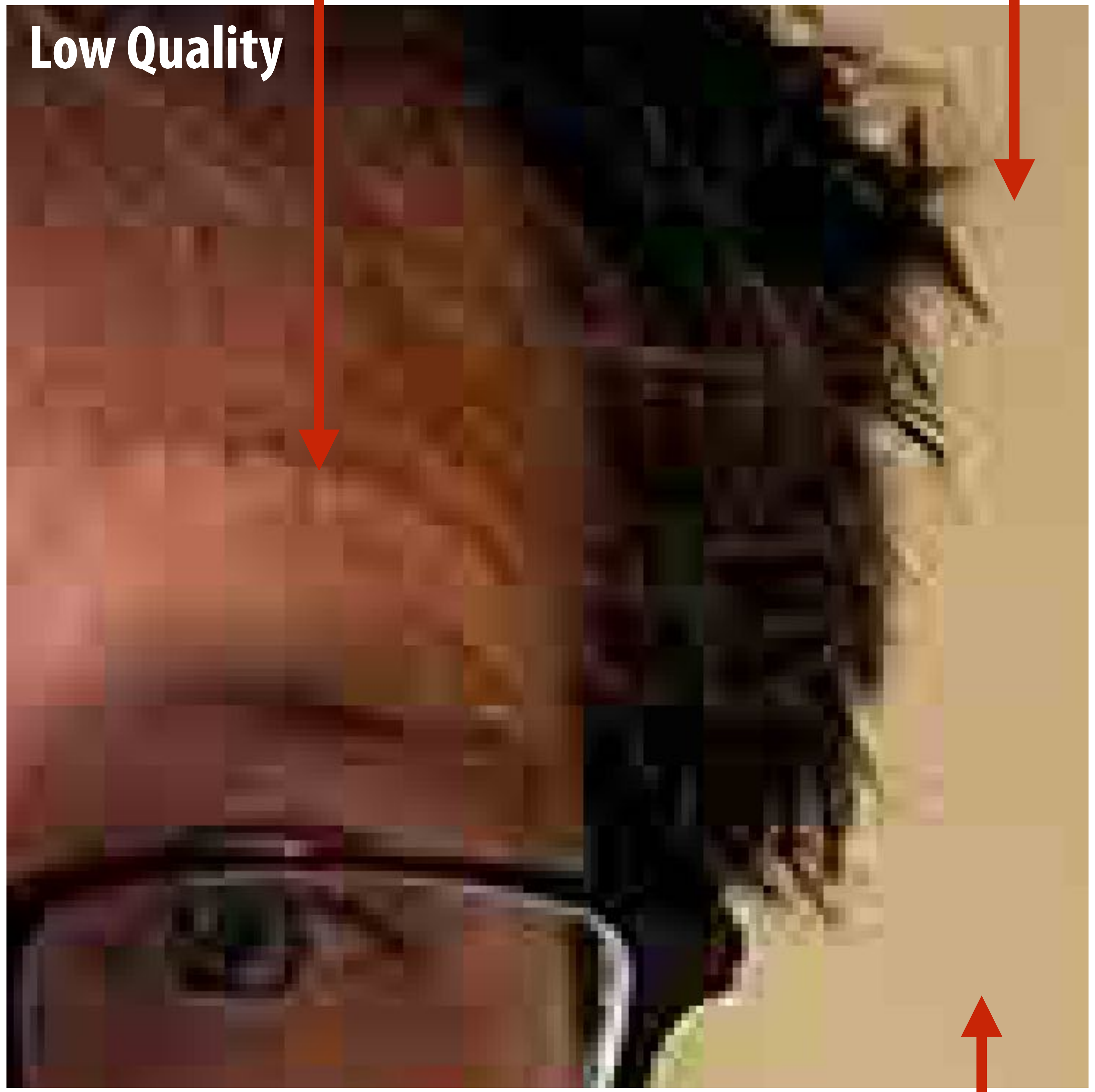
Quantization zeros out many coefficients

[Credit: Wikipedia, Pat Hanrahan]

JPEG compression artifacts

Noticeable 8x8 pixel block boundaries

Noticeable error near high gradients



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

JPEG compression artifacts

Original Image (actual size)

Original Image

Quality Level 9

Quality Level 6

Quality Level 3

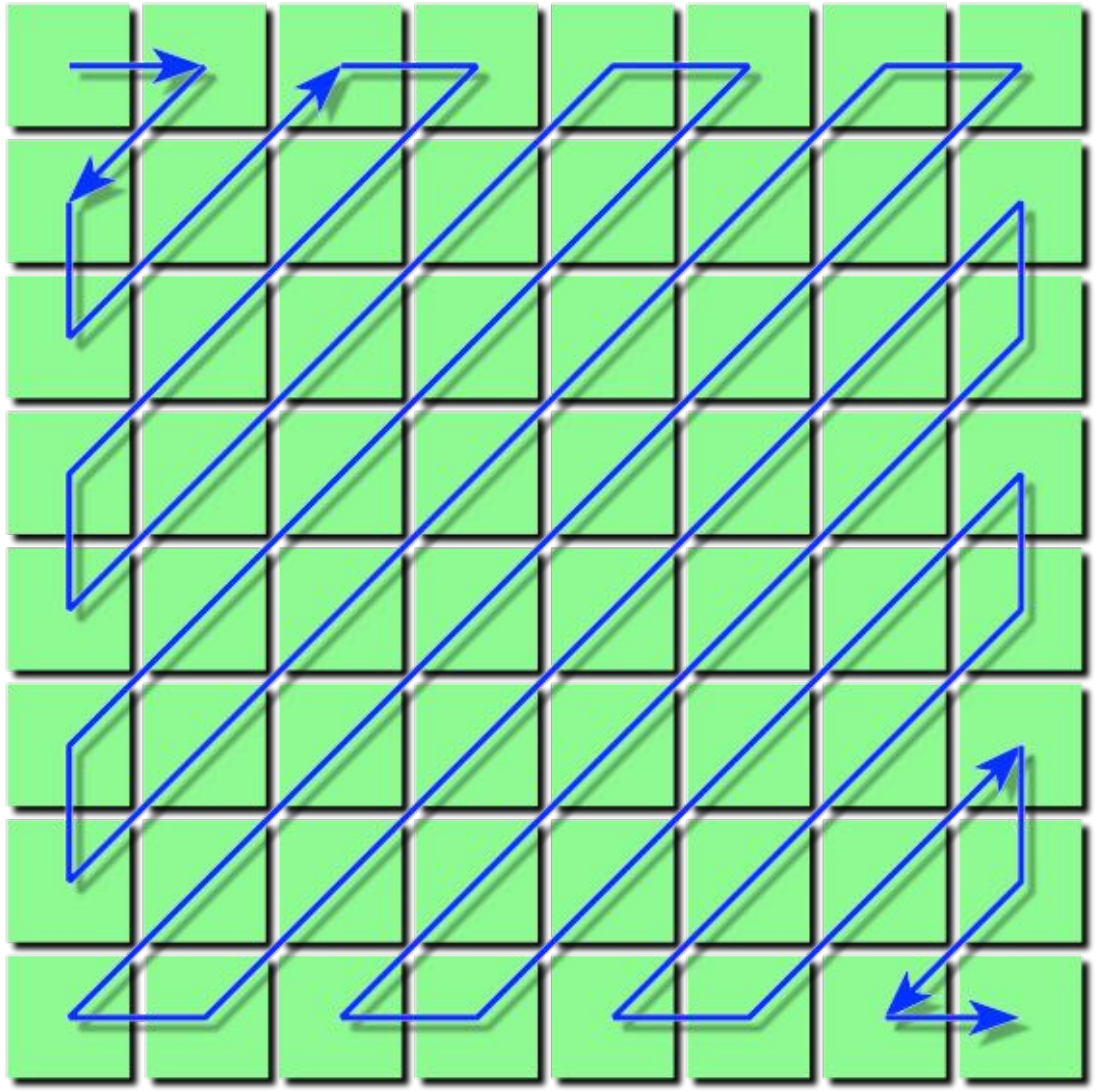
Quality Level 1

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

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

Image credit: Wikipedia

JPEG compression summary

Convert image to Y'CbCr

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

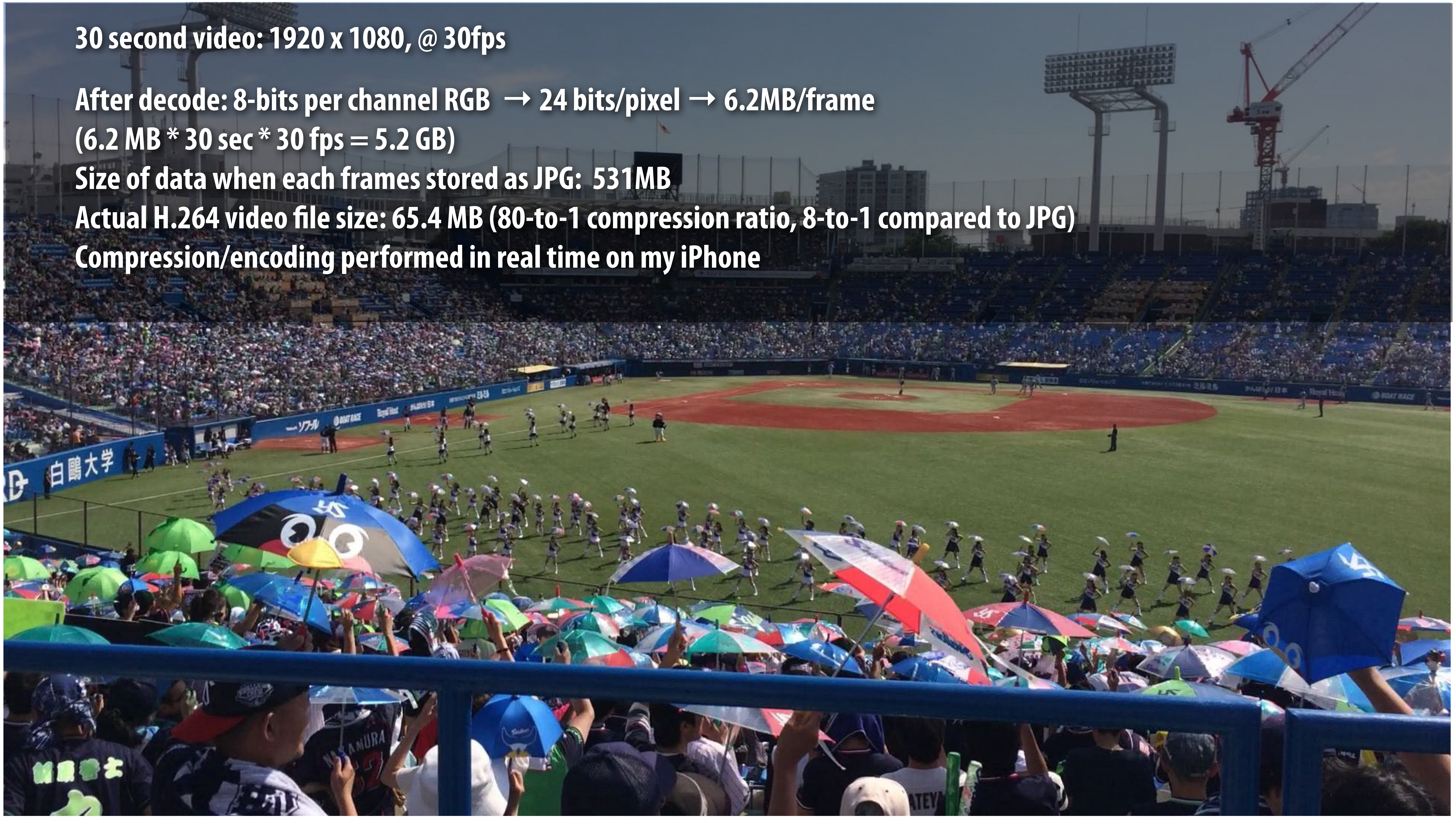
For each color channel (Y', Cb, Cr):

- 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

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: 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 most 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

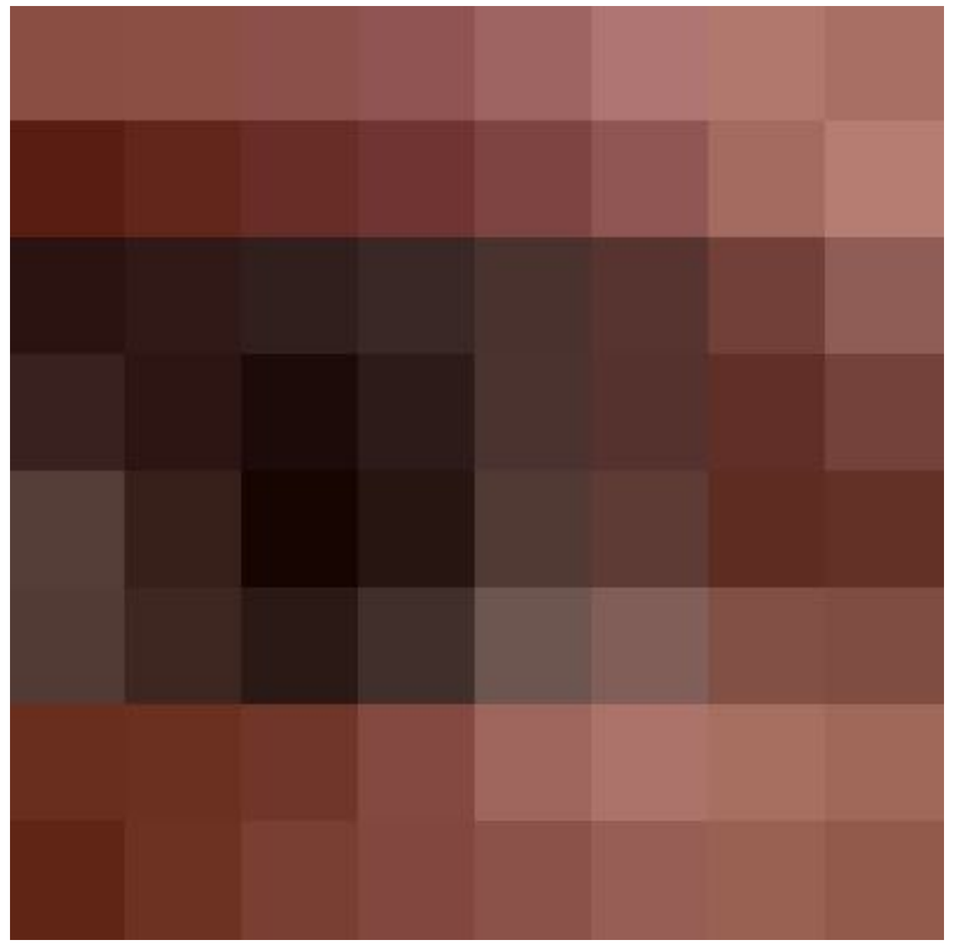
- **Video container (MOV, AVI) bundles media assets**

- **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

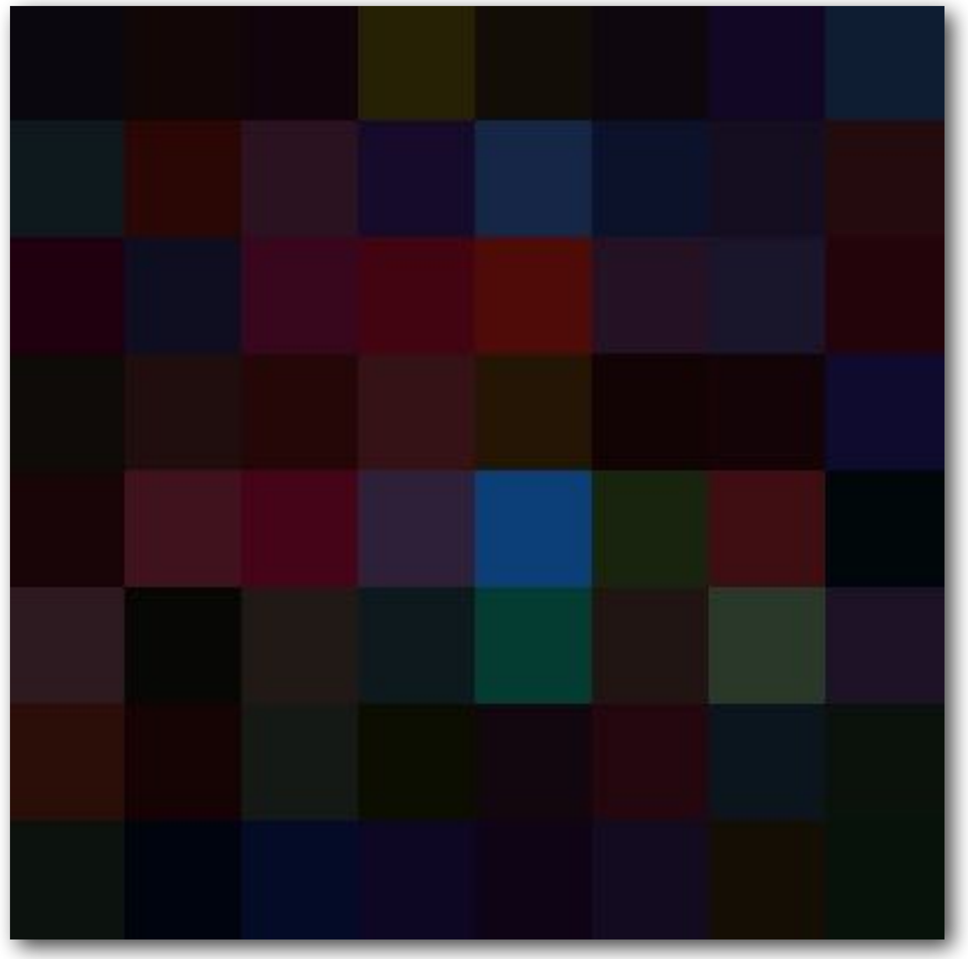
Video compression: main ideas

- **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)**

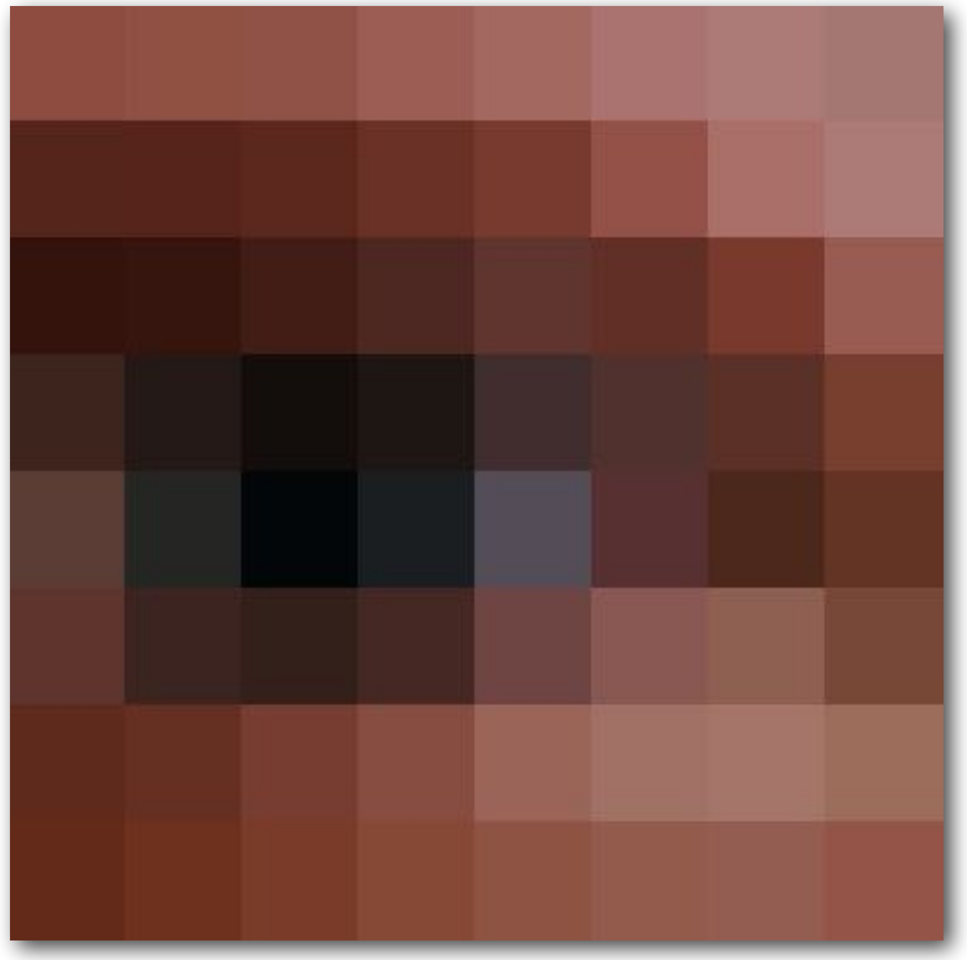
Residual: difference between compressed image and original image



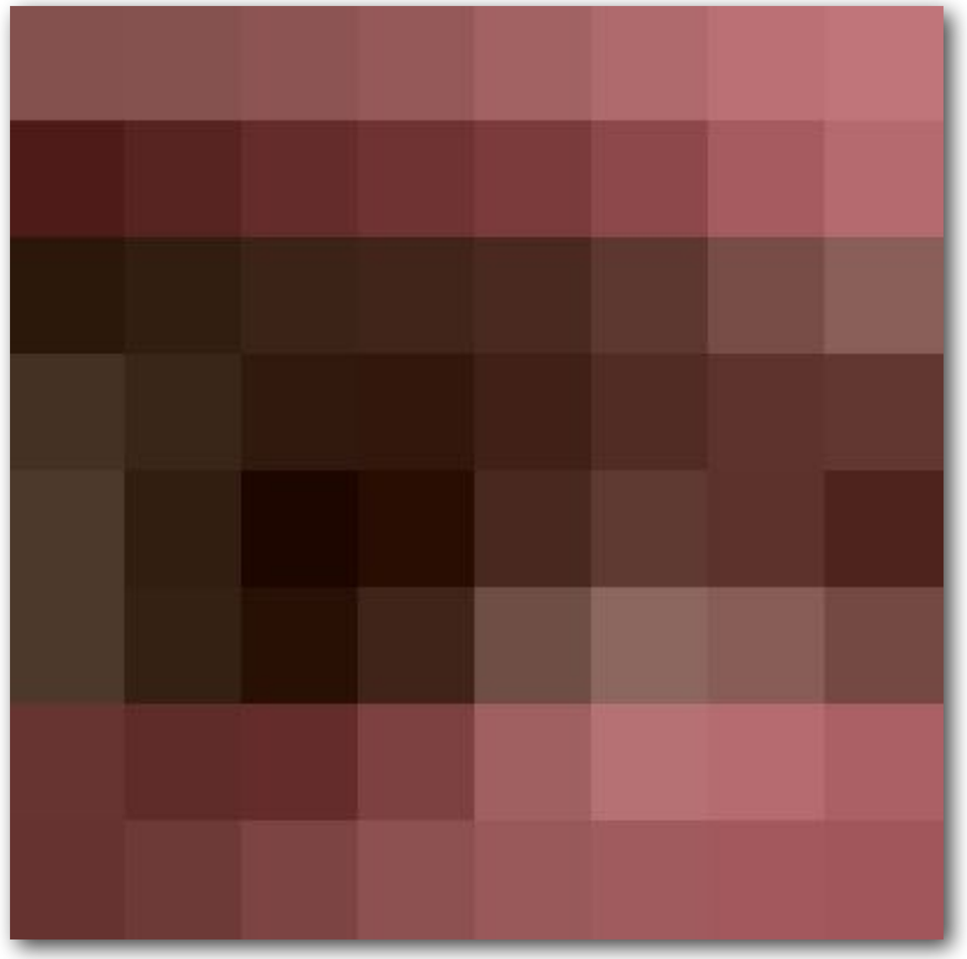
Compressed pixels
(JPEG quality level 6)



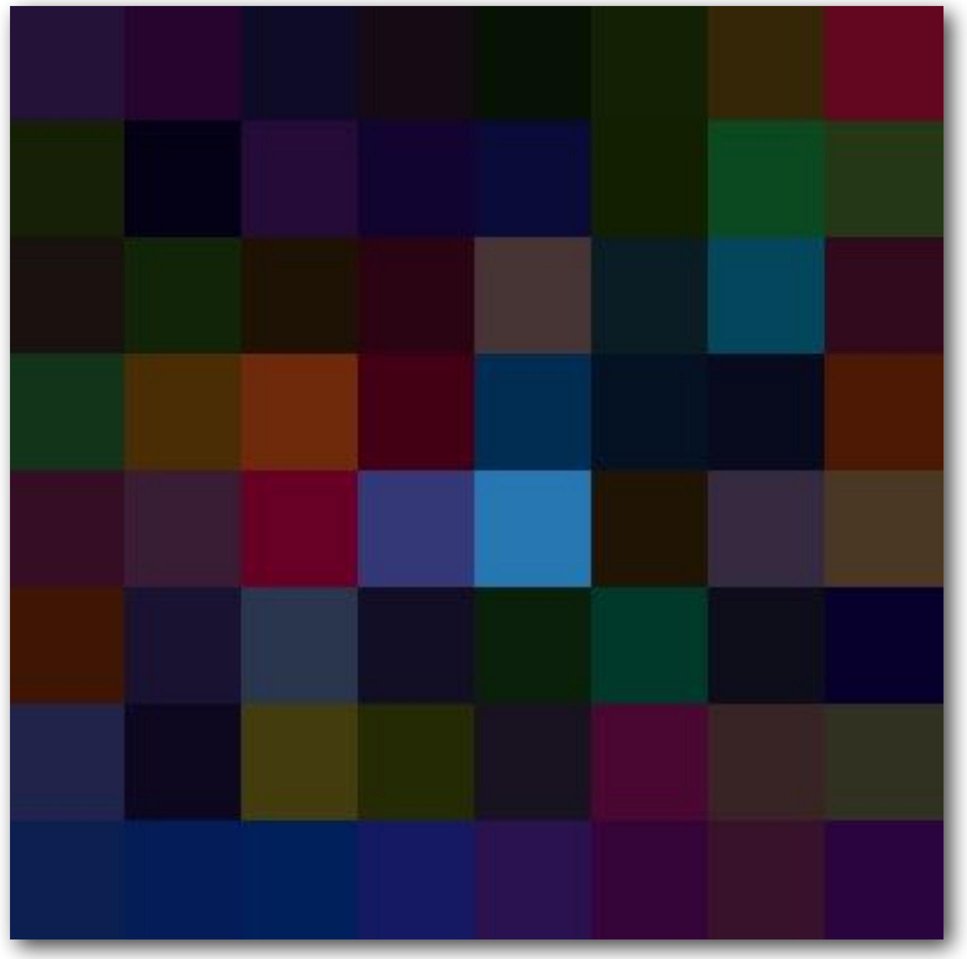
Residual
(amplified for visualization)



Original pixels



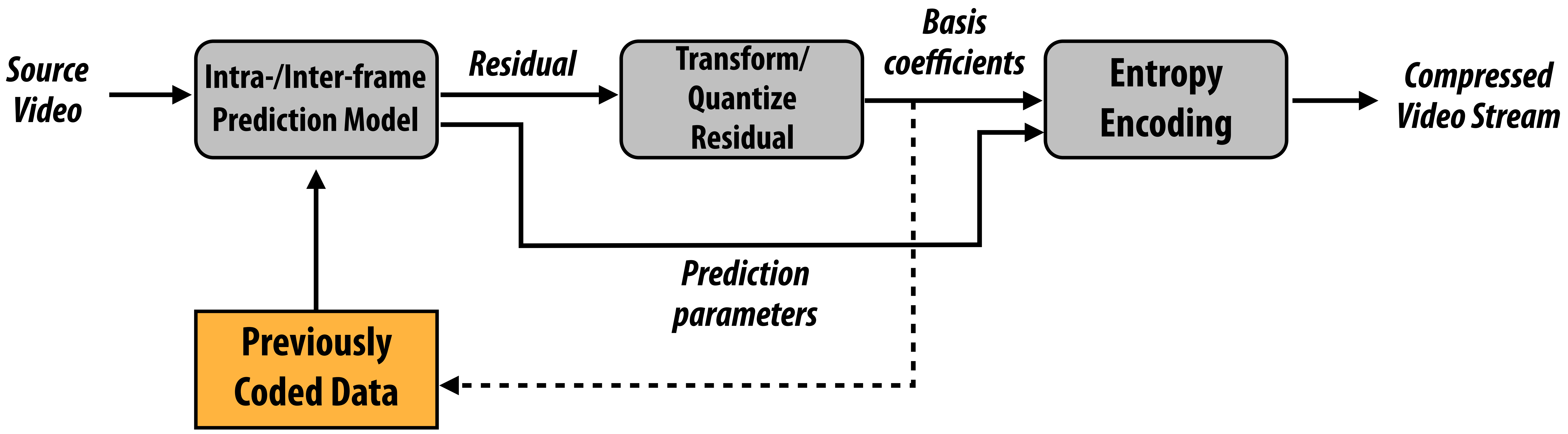
Compressed pixels
(JPEG quality level 2)



Residual
(amplified for visualization)

In video compression schemes, the residual image is compressed using lossy compression techniques like those described in the earlier part of this lecture. Better predictions lead to smaller and more compressible residuals!

H.264/AVC video compression overview

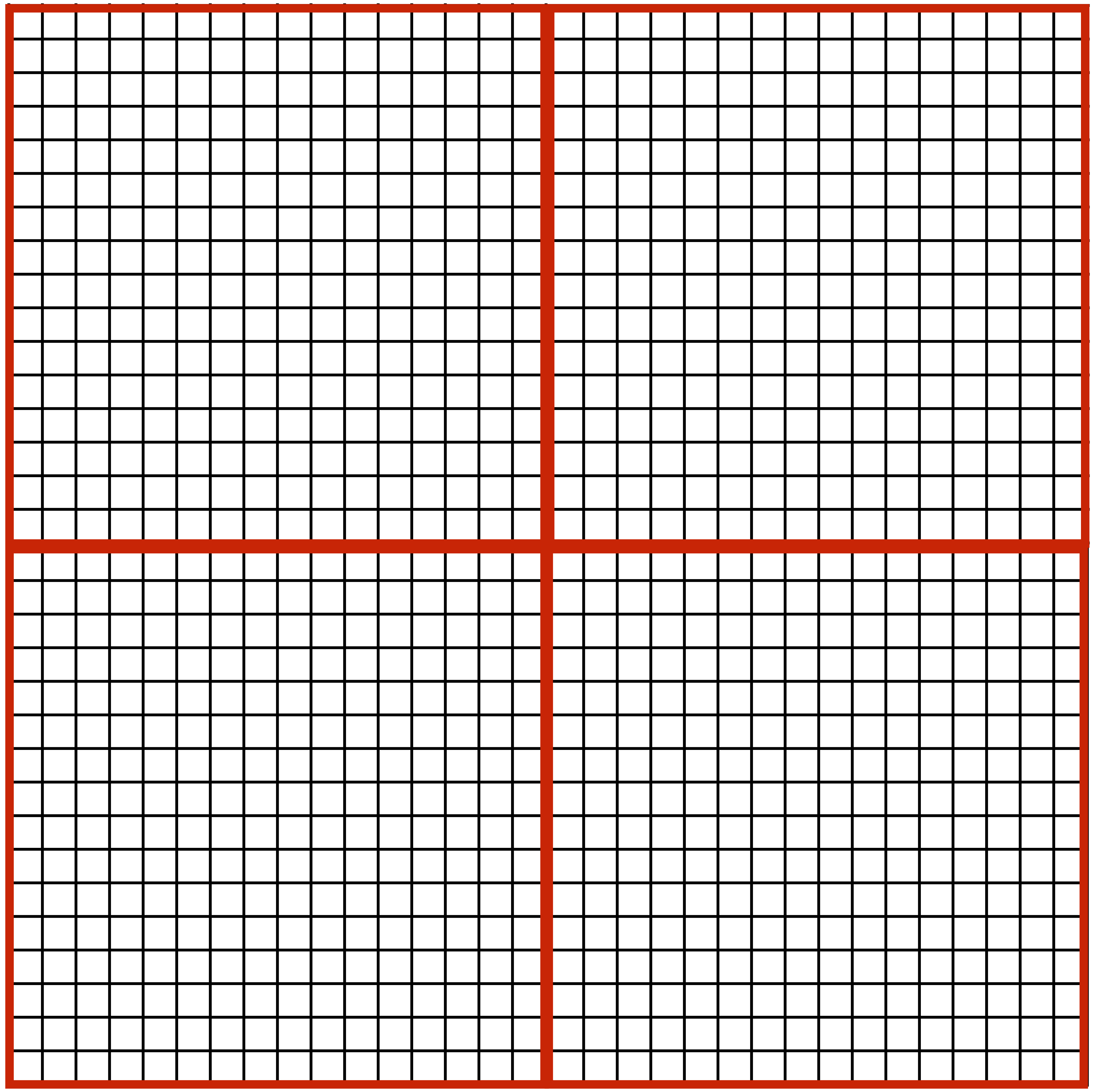


Residual: difference between predicted pixel values and input video pixel values

In other words: The main idea today: use an algorithm to predict what a future pixel should be, then store a description of the algorithm and the residual of the prediction.

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

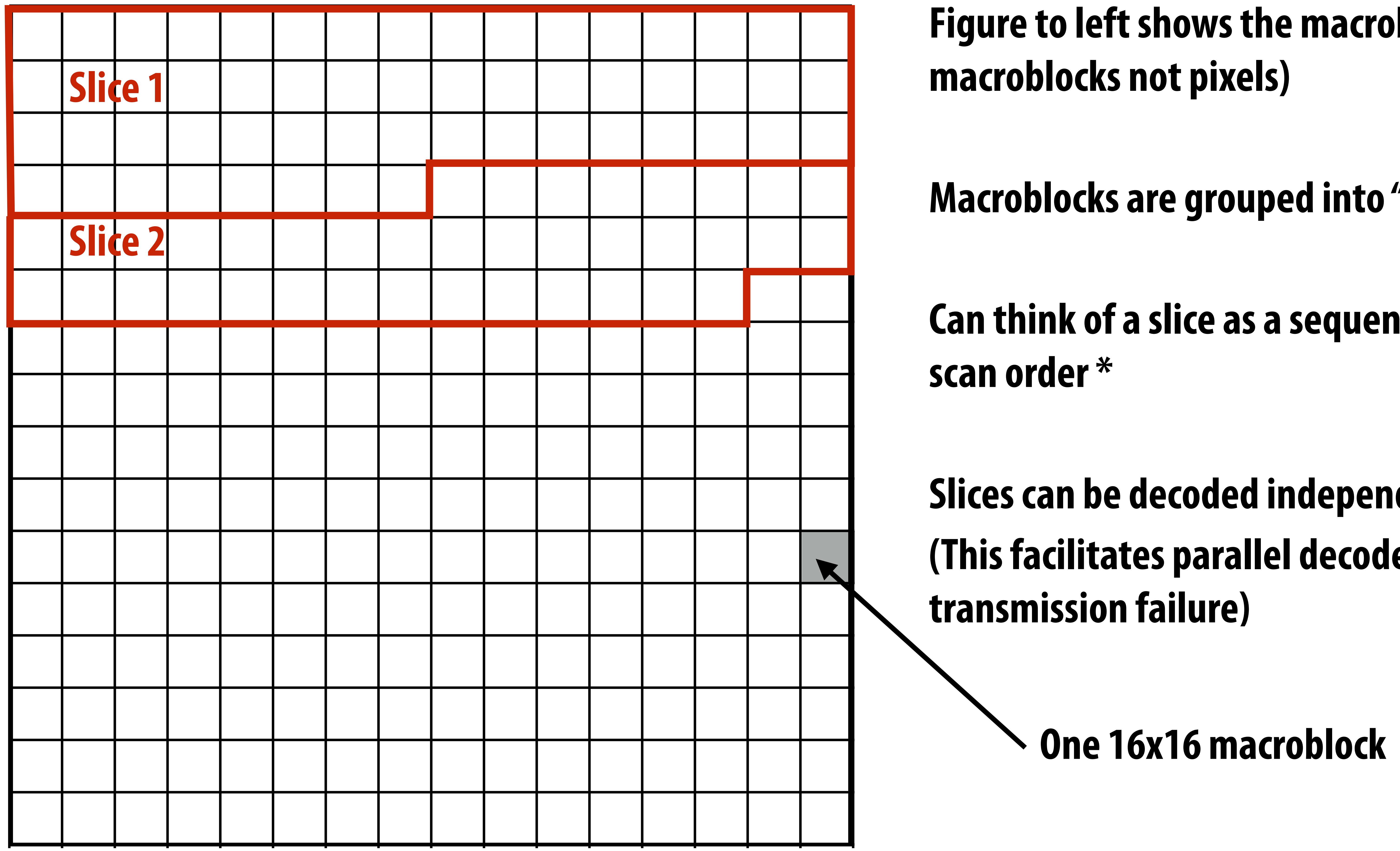


Figure to left shows the macroblocks in a frame (boxes are macroblocks not pixels)

Macroblocks are grouped into "slices"

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

Slices can be decoded independently **

(This facilitates parallel decode or robustness to transmission failure)

* 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.

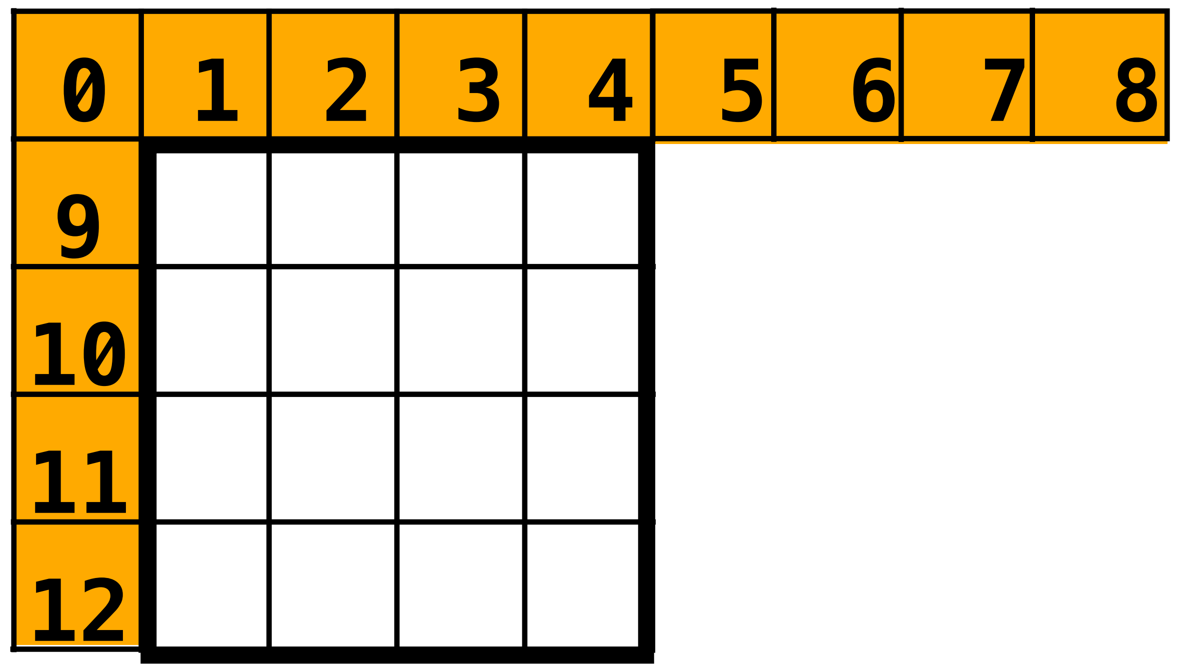
Decoding via prediction + correction

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

- **Three forms of prediction:**
  - **I-macroblock: ("intra-picture predictive only") macroblock samples predicted from samples in previous macroblocks in the same slice of the current frame**
  - **P-macroblock: ("predictive") macroblock pixel samples can be predicted from samples from one other frame (one prediction per macroblock)**
  - **B-macroblock: ("bipredictive") macroblock pixel samples can be predicted by a weighted combination of multiple predictions from samples from other frames**

Intra-frame prediction (I-macroblock)

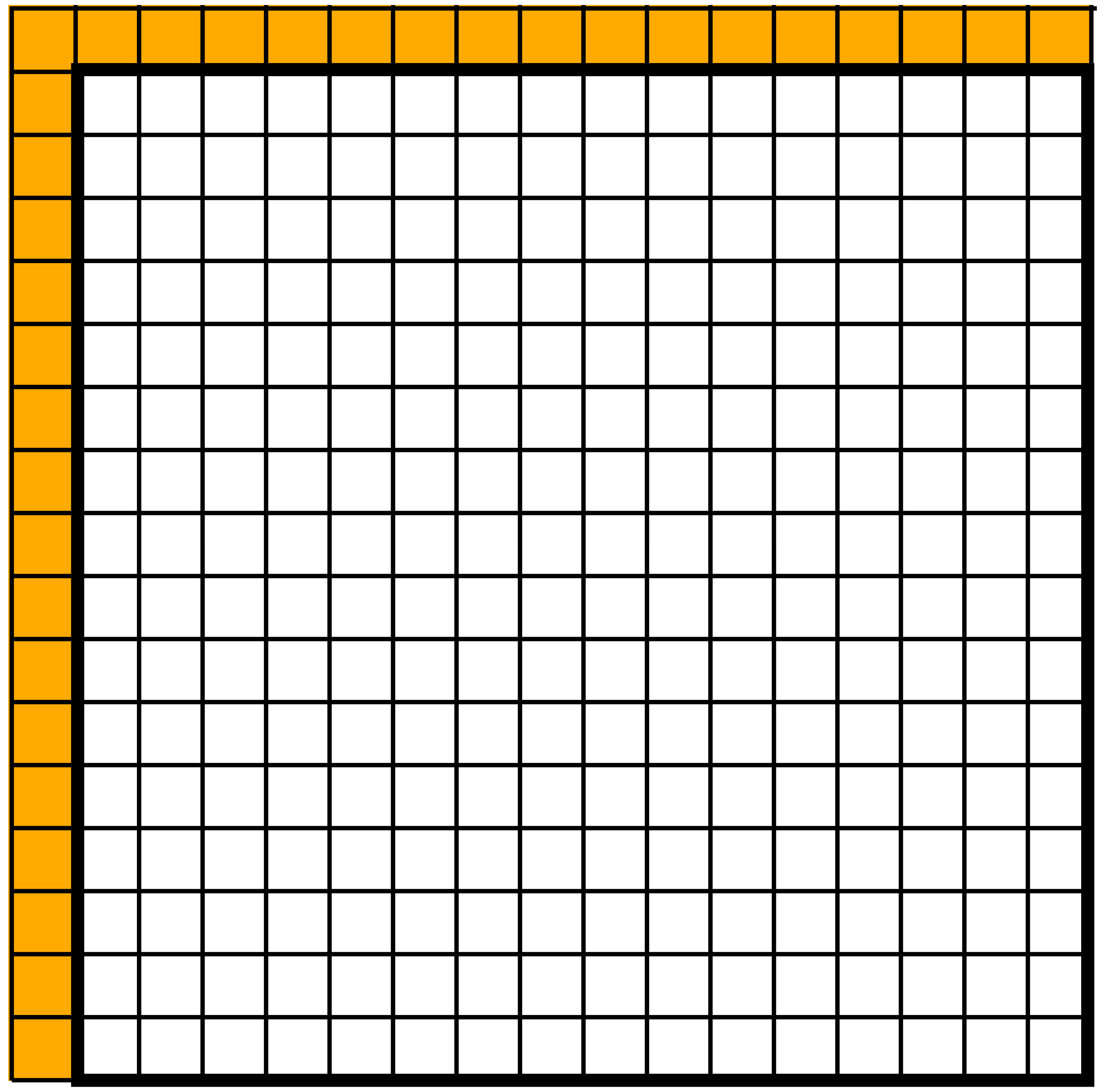
- Prediction of sample values is performed in spatial domain, not transform domain
  - Predict pixel values, not basis coefficients
- Modes for predicting the 16x16 luma (Y) values: *
  - Intra_4x4 mode: predict 4x4 block of samples from adjacent row/col of pixels
  - Intra_16x16 mode: predict entire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

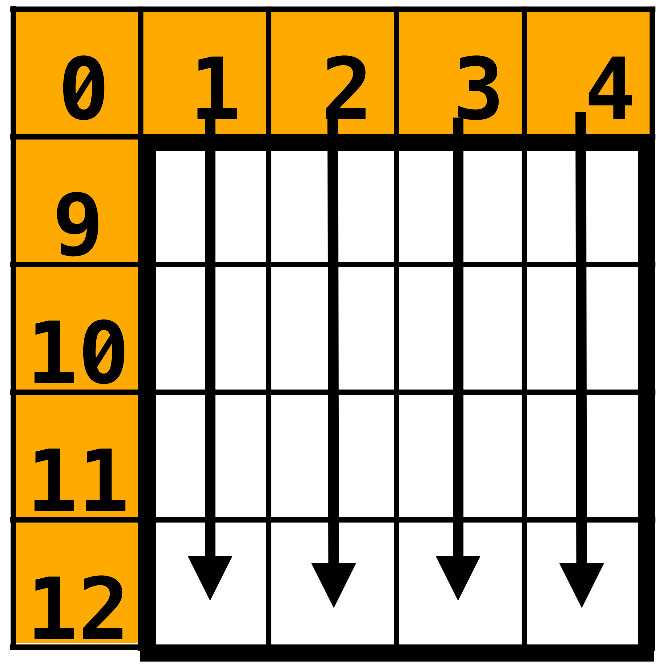


Intra_16x16

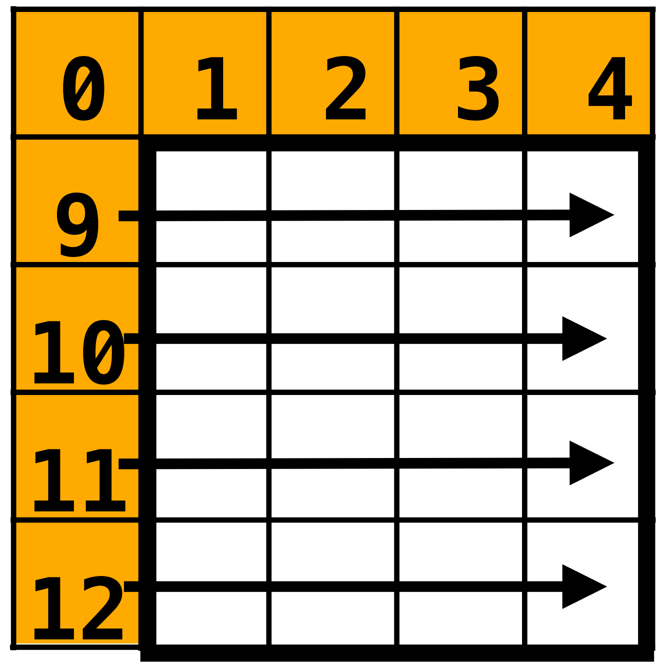
* An additional 8x8 mode exists in the H.264 High Profile

Intra_4x4 prediction modes

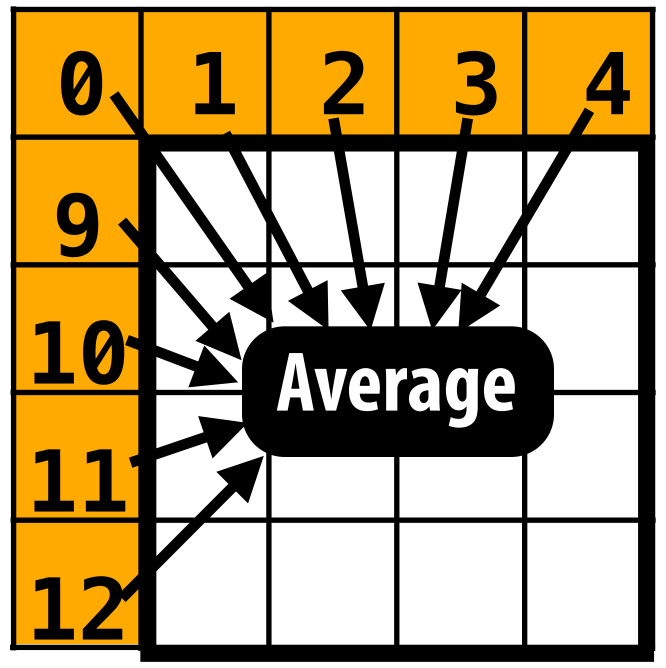
- Nine prediction modes (6 shown below)
  - Other modes: horiz-down, vertical-left, horiz-up



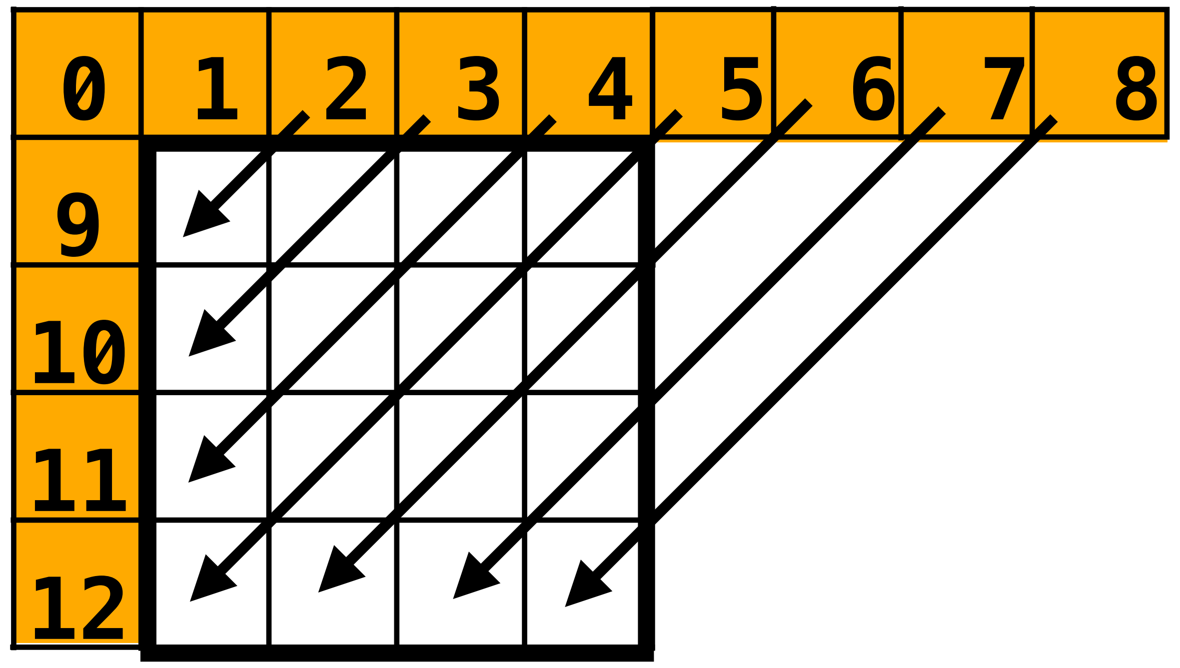
Mode 0: vertical
(4x4 block is copy of above row of pixels)



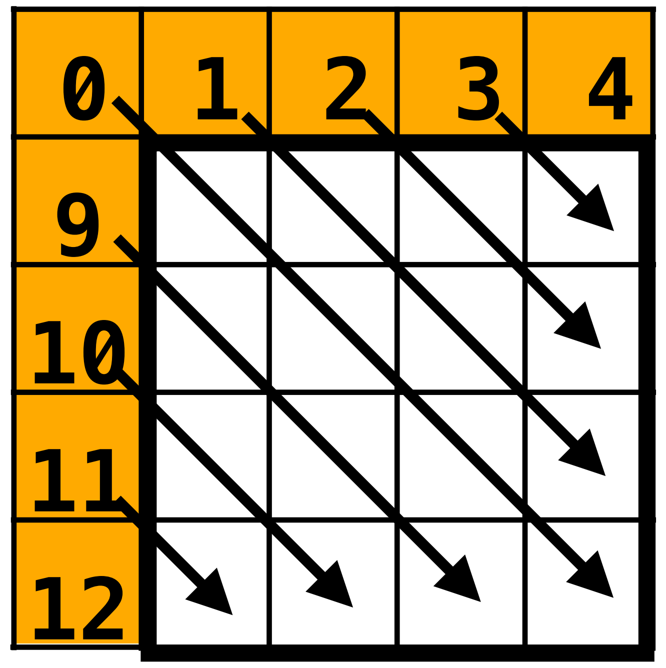
Mode 1: horizontal
(4x4 block is copy of left col of pixels)



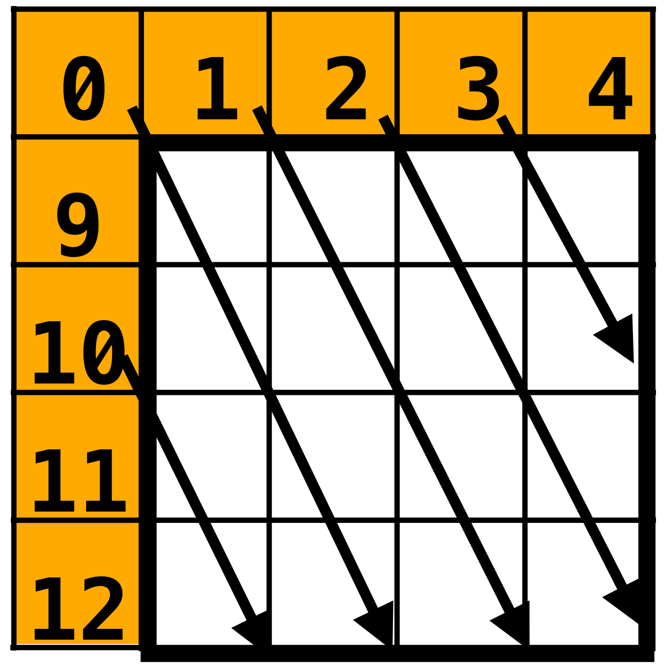
Mode 2: DC
(4x4 block is average of above row and left col of pixels)



Mode 3: diagonal down-left (45°)

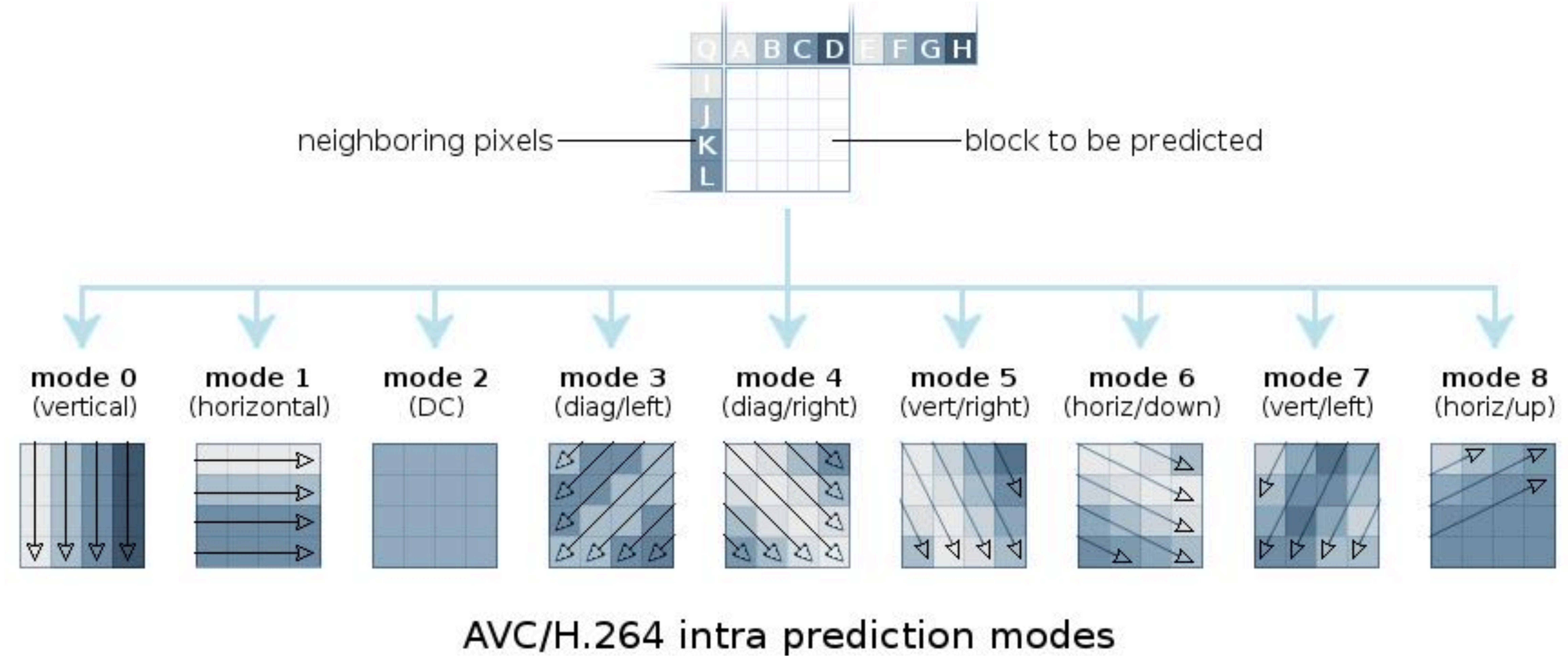


Mode 4: diagonal down-right (45°)



Mode 5: vertical-right (26.6°)

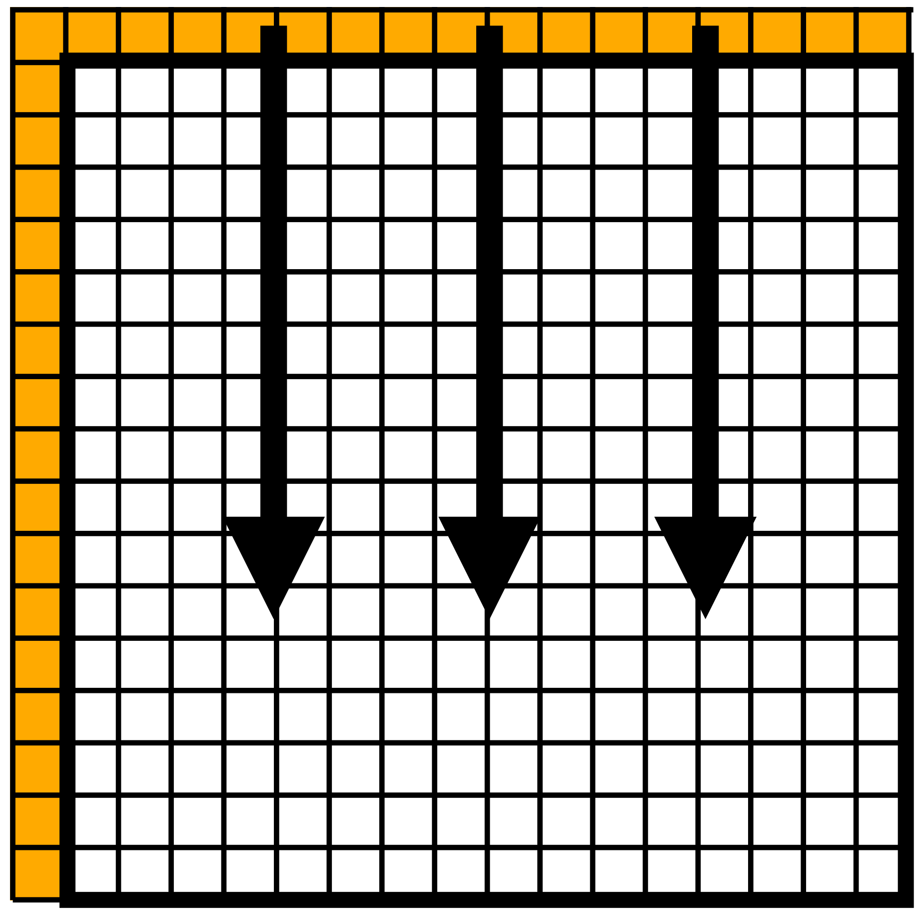
Intra_4x4 prediction modes (another look)



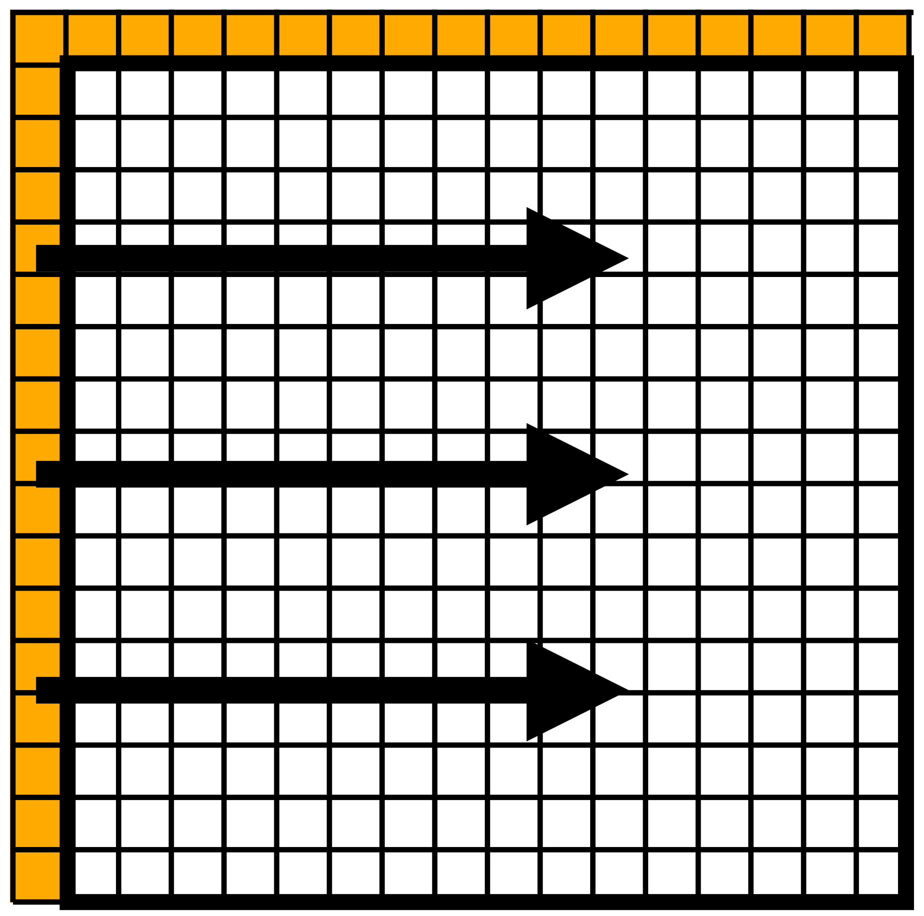
https://hacks.mozilla.org/2018/06/av1-next-generation-video-the-constrained-directional-enhancement-filter/

Intra_16x16 prediction modes

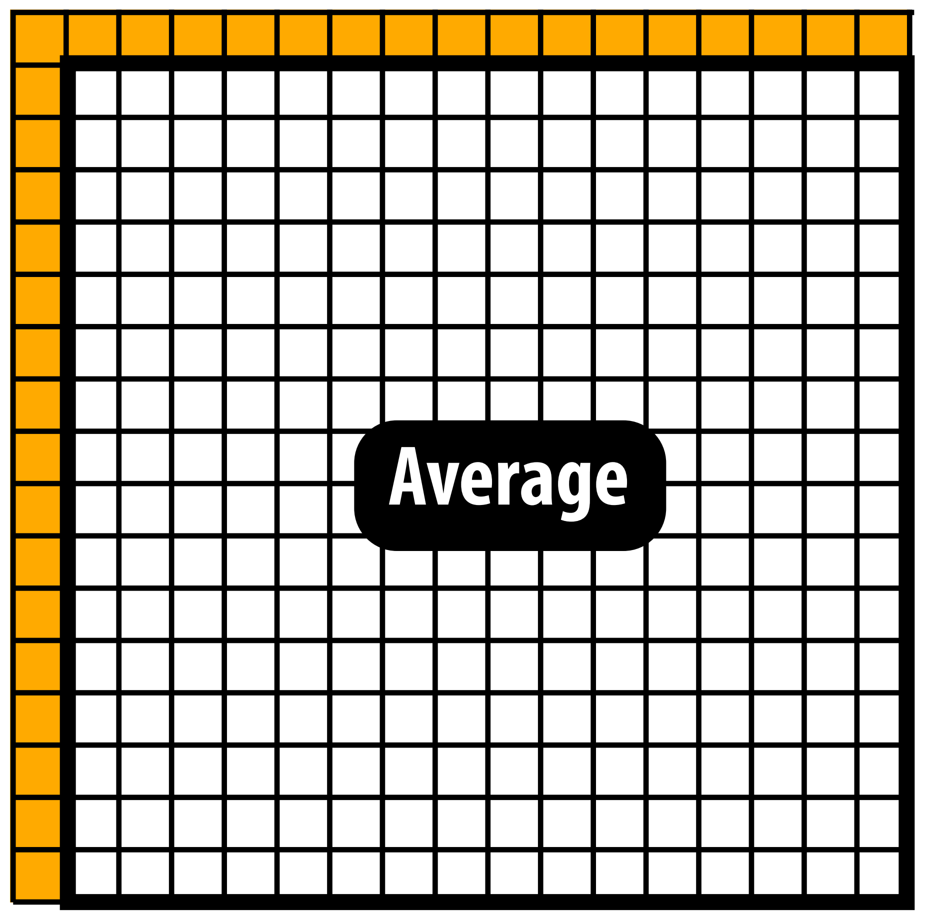
4 prediction modes: vertical, horizontal, DC, plane



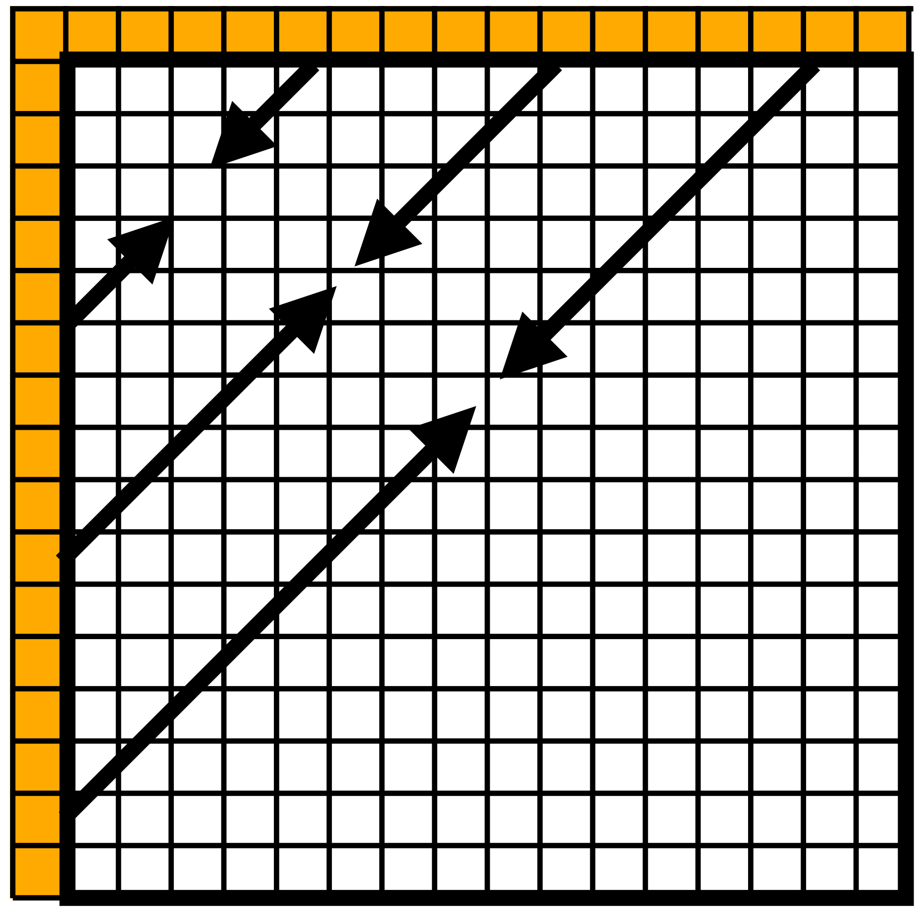
Mode 0: vertical



Mode 1: horizontal



Mode 2: DC



Mode 4: plane

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

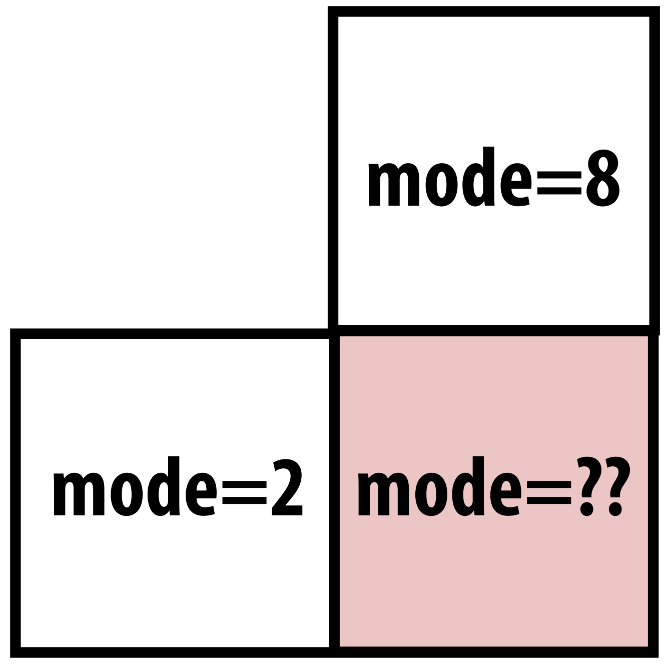
A derived from top row, B derived from left col, C from both

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)

Each mode is a different prediction algorithm, so we have to store which algorithm we chose in the video stream in order to decode it.

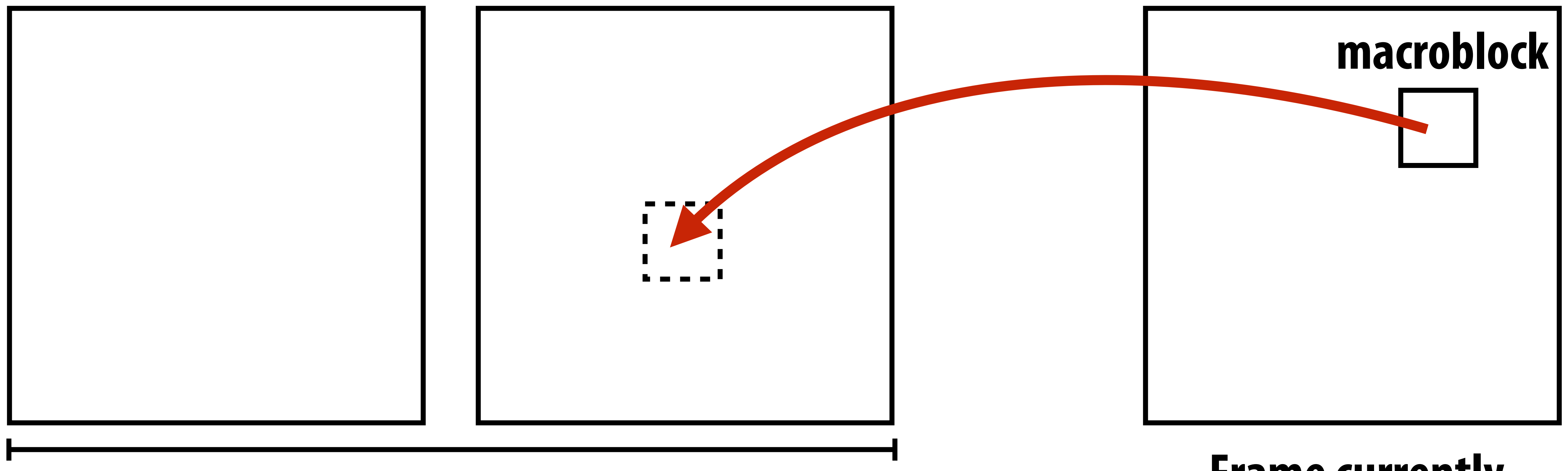
- Intra-prediction scheme for each 4x4 block within macroblock encoded as follows:
  - One bit per 4x4 block:
    - if 1, use <u>most probable</u> mode
      - Most probable = lower of modes used for 4x4 block to left or above current block
    - if 0, use additional 3-bit value `rem_intra4x4_pred_mode` to encode one of nine modes
      - if `intra4x4_pred_mode` is smaller than most probable mode, then actual mode is given by `intra4x4_pred_mode`
      - else, actual mode is `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



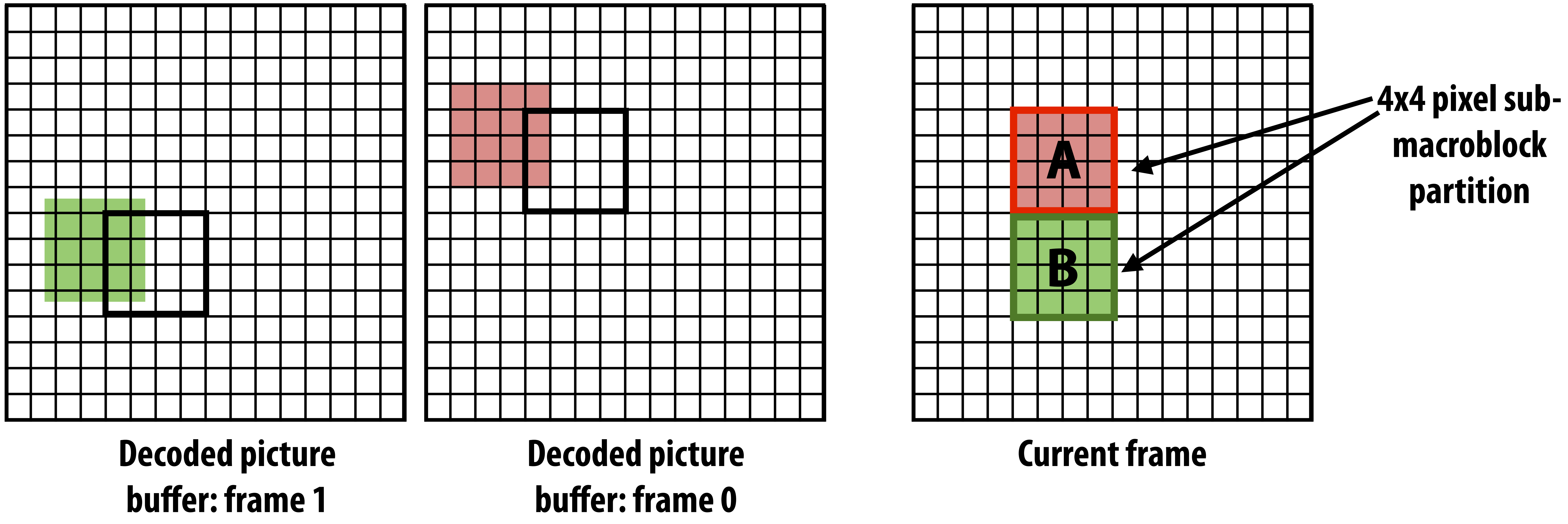
Recently decoded frames
(stored in "decoded picture buffer")

Frame currently
being decoded

* 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

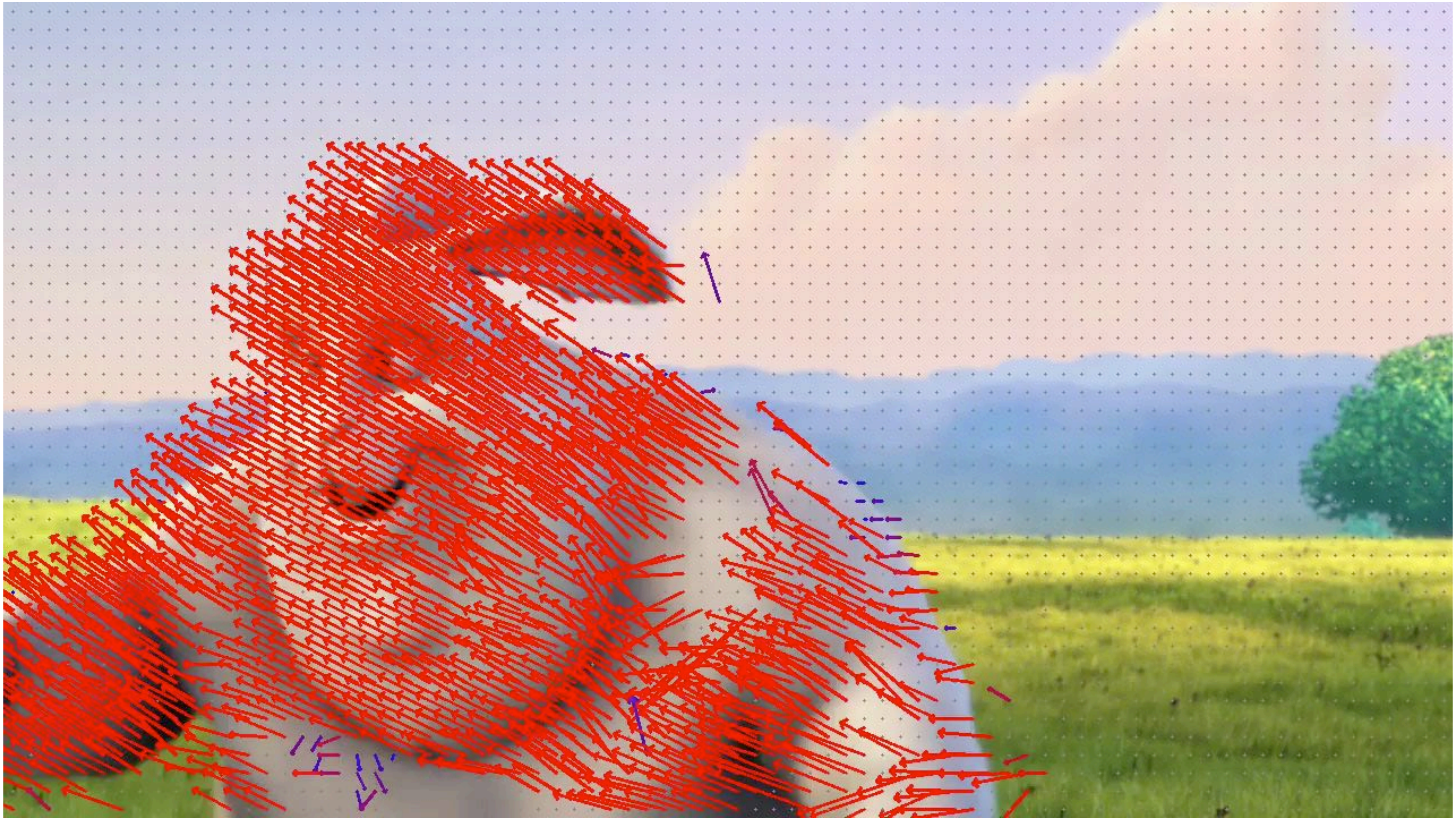
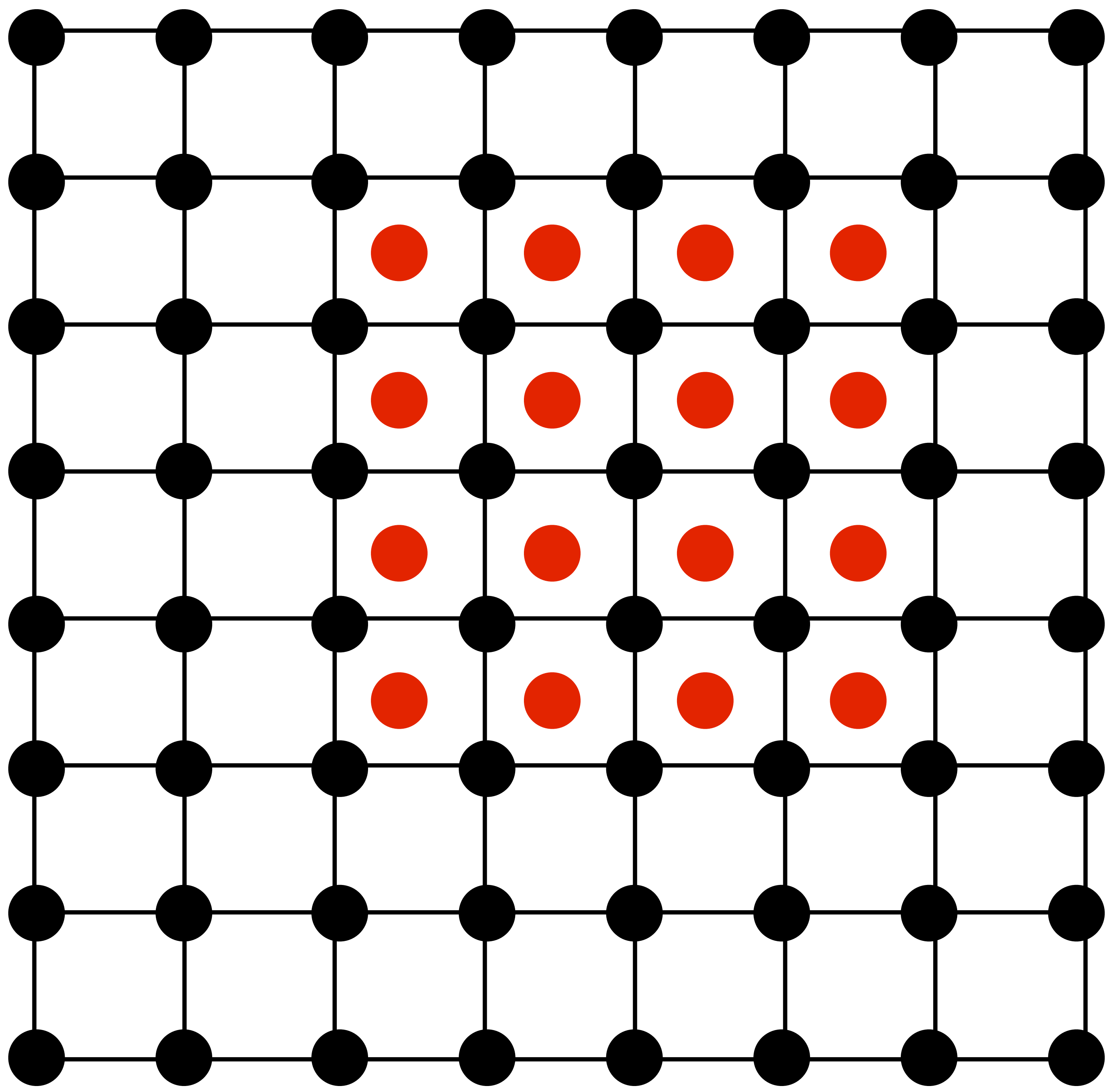
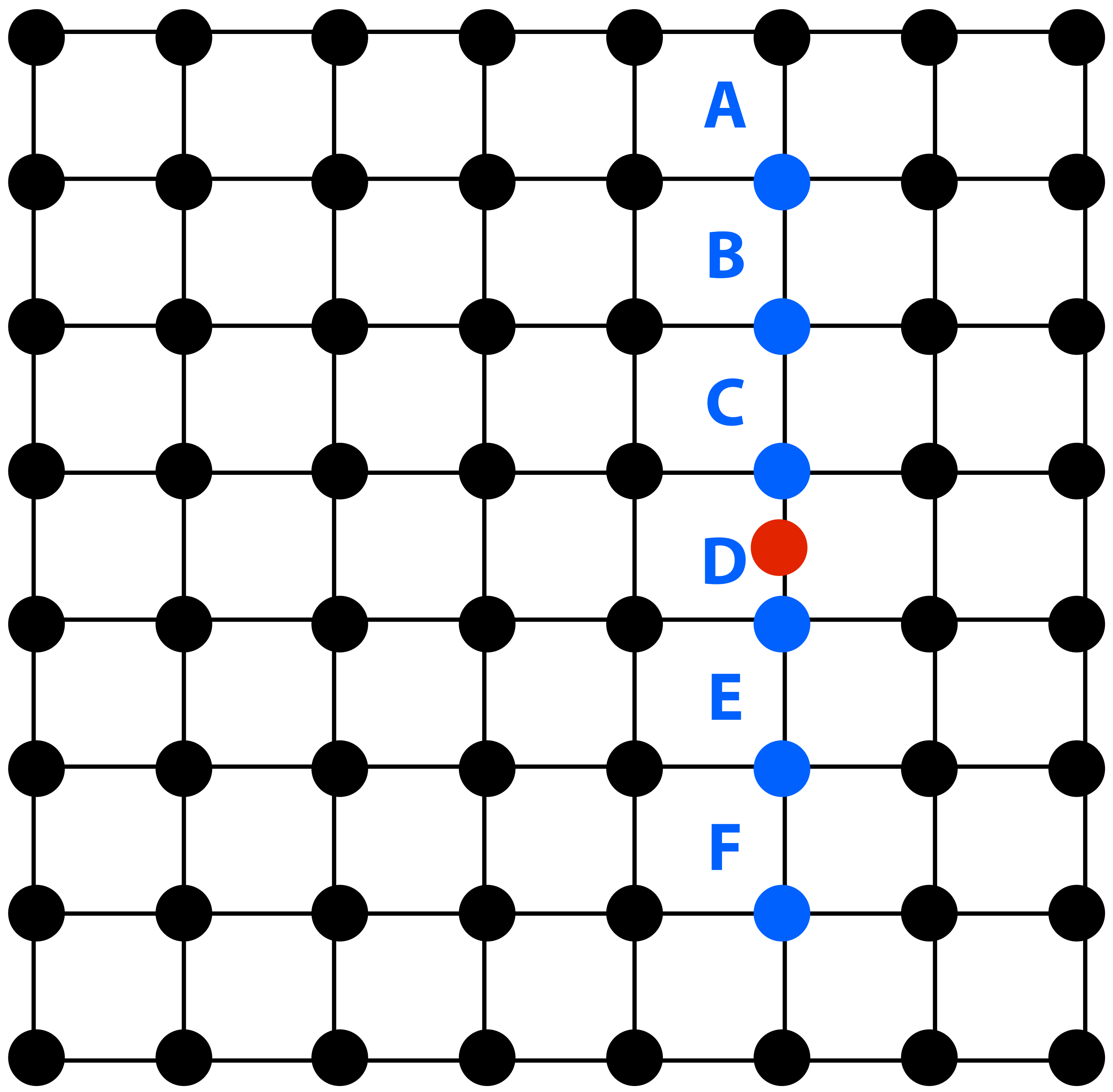


Image credit: Keyi Zhang

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)

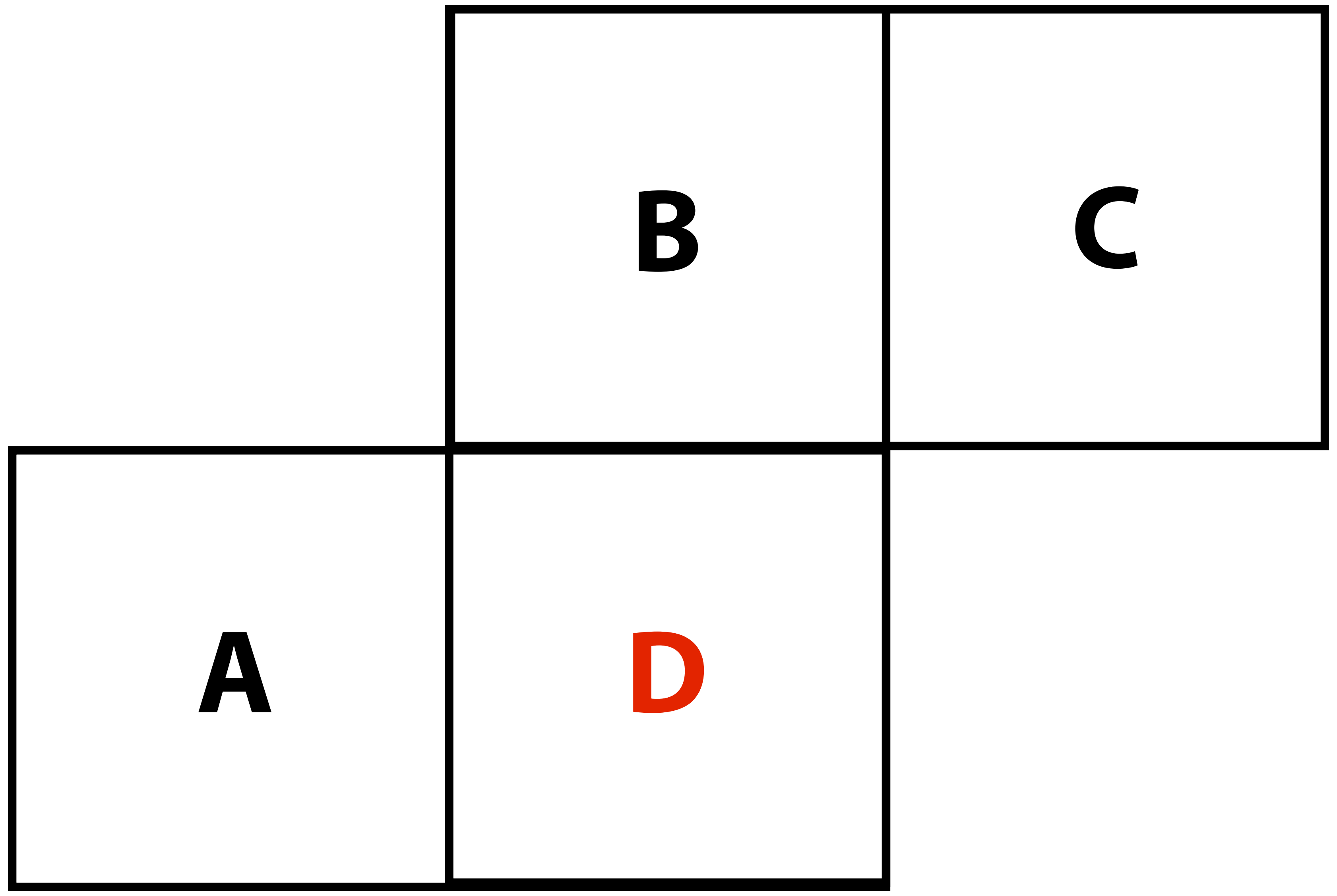
H.264 supports both 1/2 pixel and 1/4 pixel resolution motion vectors

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 partitions of neighboring blocks are of different size



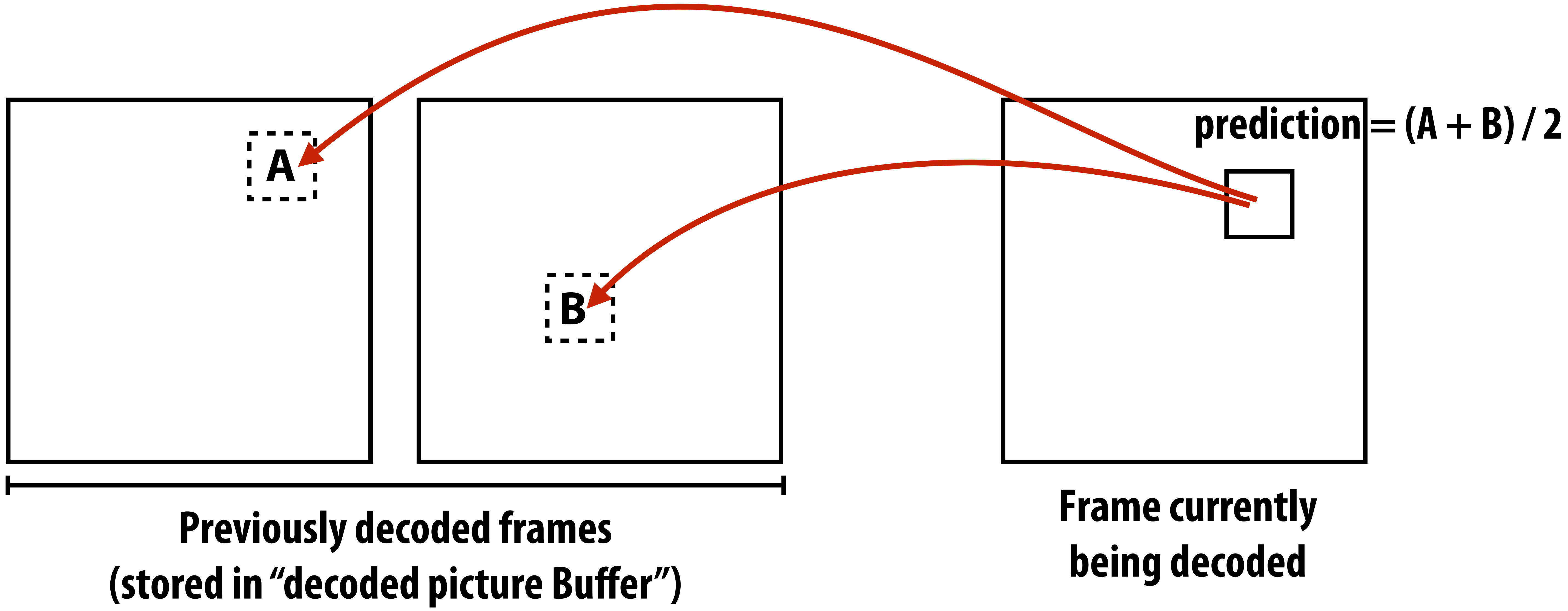
Question: what partition size is best?

- 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)
  - Must store:
    - source 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)

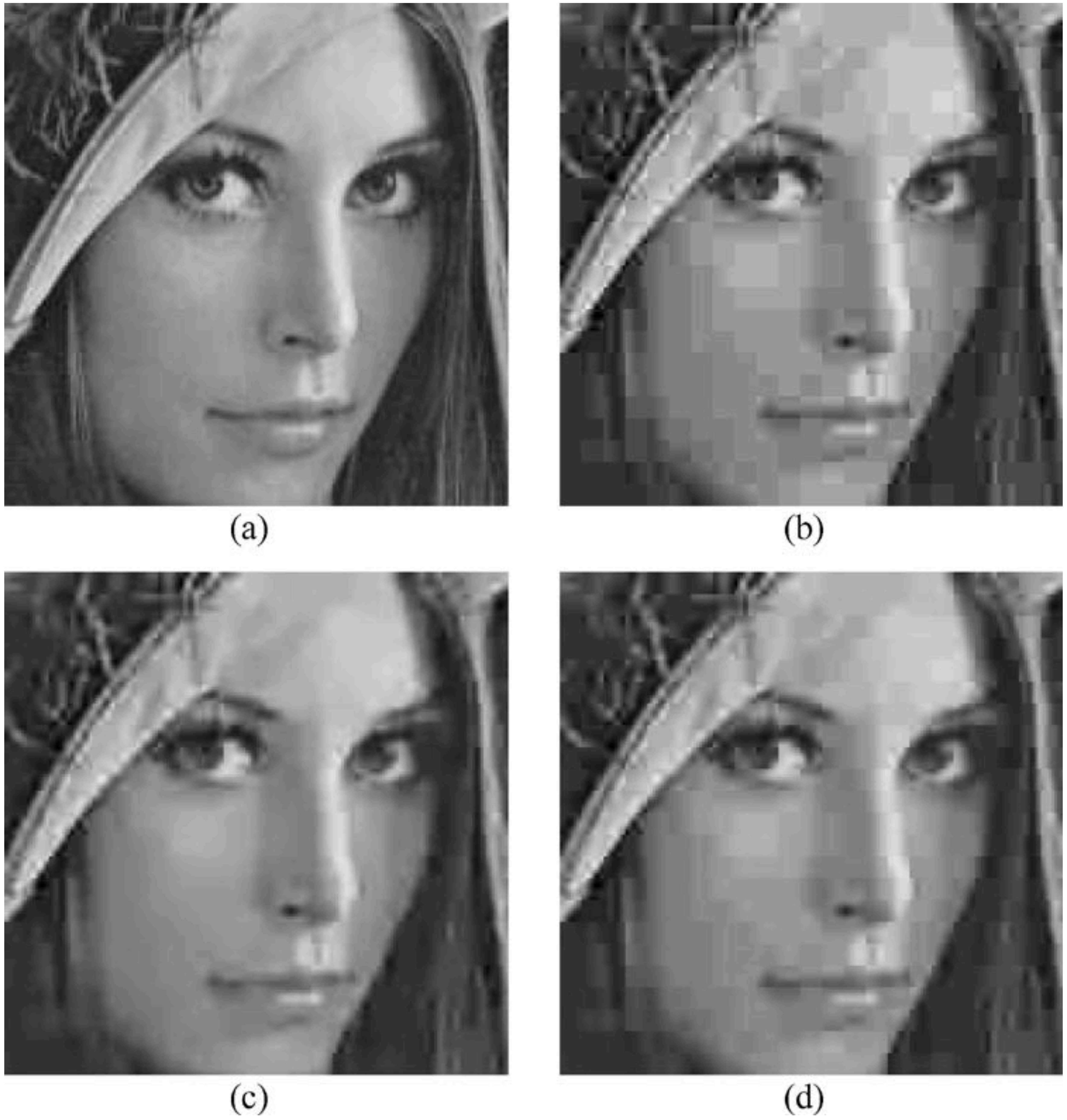


Additional prediction details

- **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

Post-process filtering

- **Deblocking**
  - Blocking artifacts may result as a result of macroblock granularity encoding
  - After macroblock decoding is complete, optionally perform smoothing filter across block edges.



[Image credit: Averbuch et al. 2005]

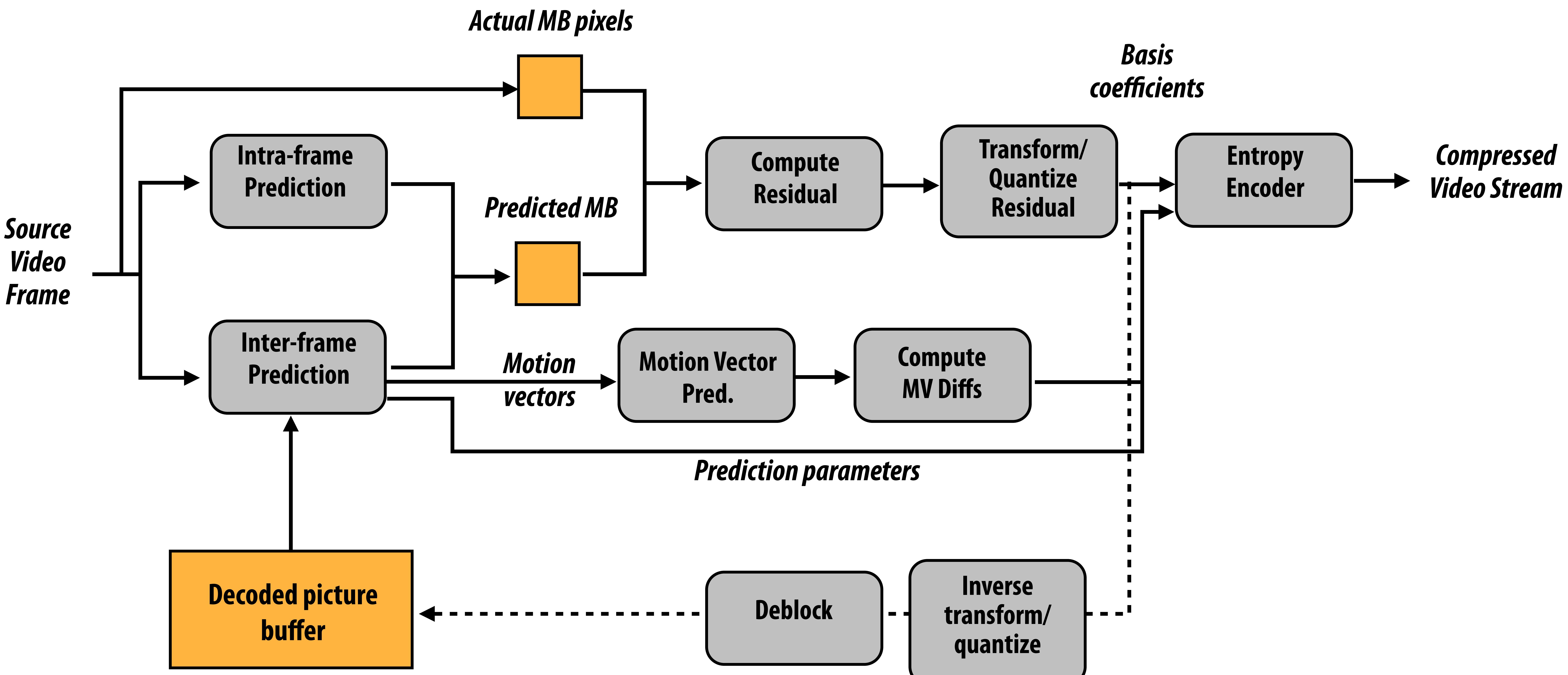
Putting it all together: encoding an inter-predicted macroblock

- Inputs:
  - Current state of decoded picture buffer (state of the video 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

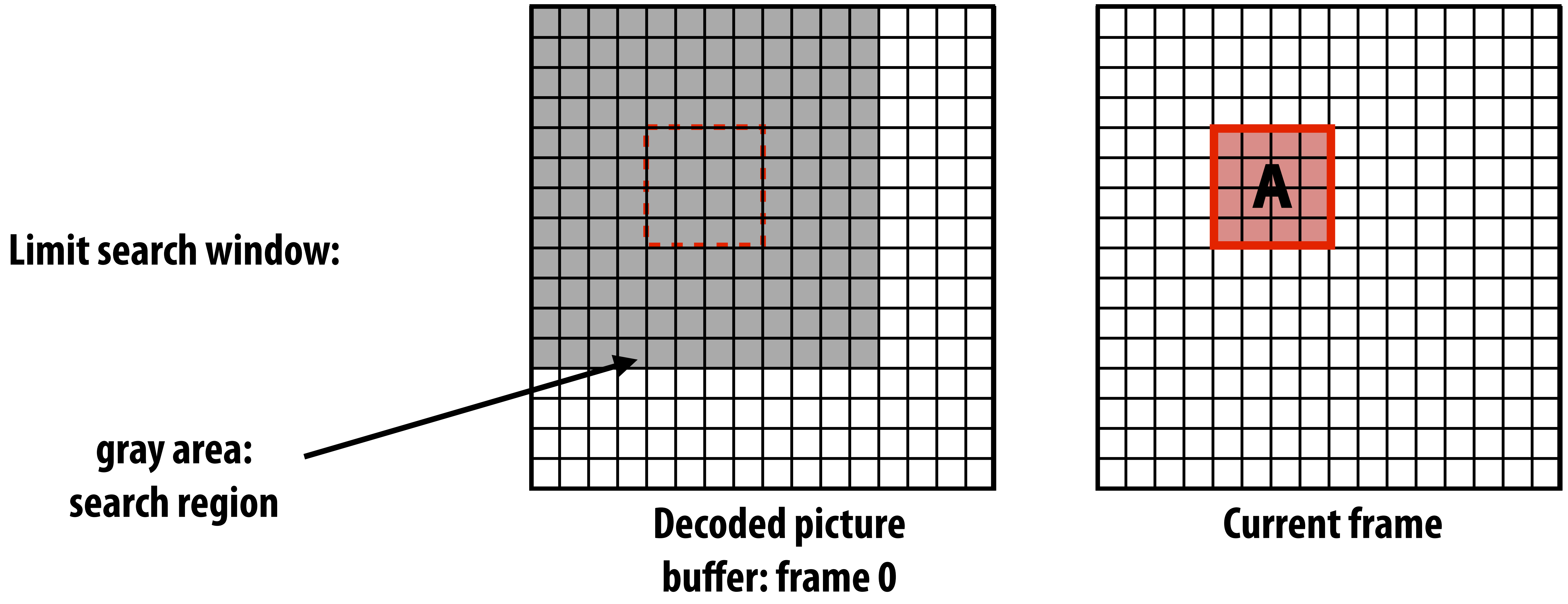
MB = macroblock
MV = motion vector



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

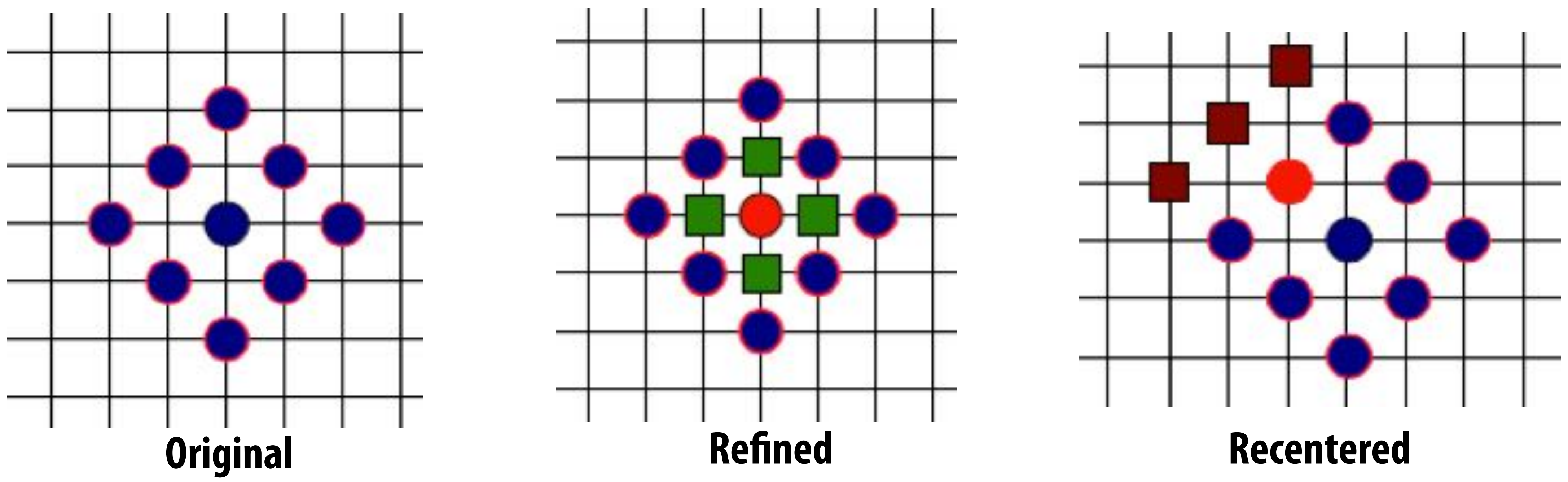
Motion estimation

- Encoder must find 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



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

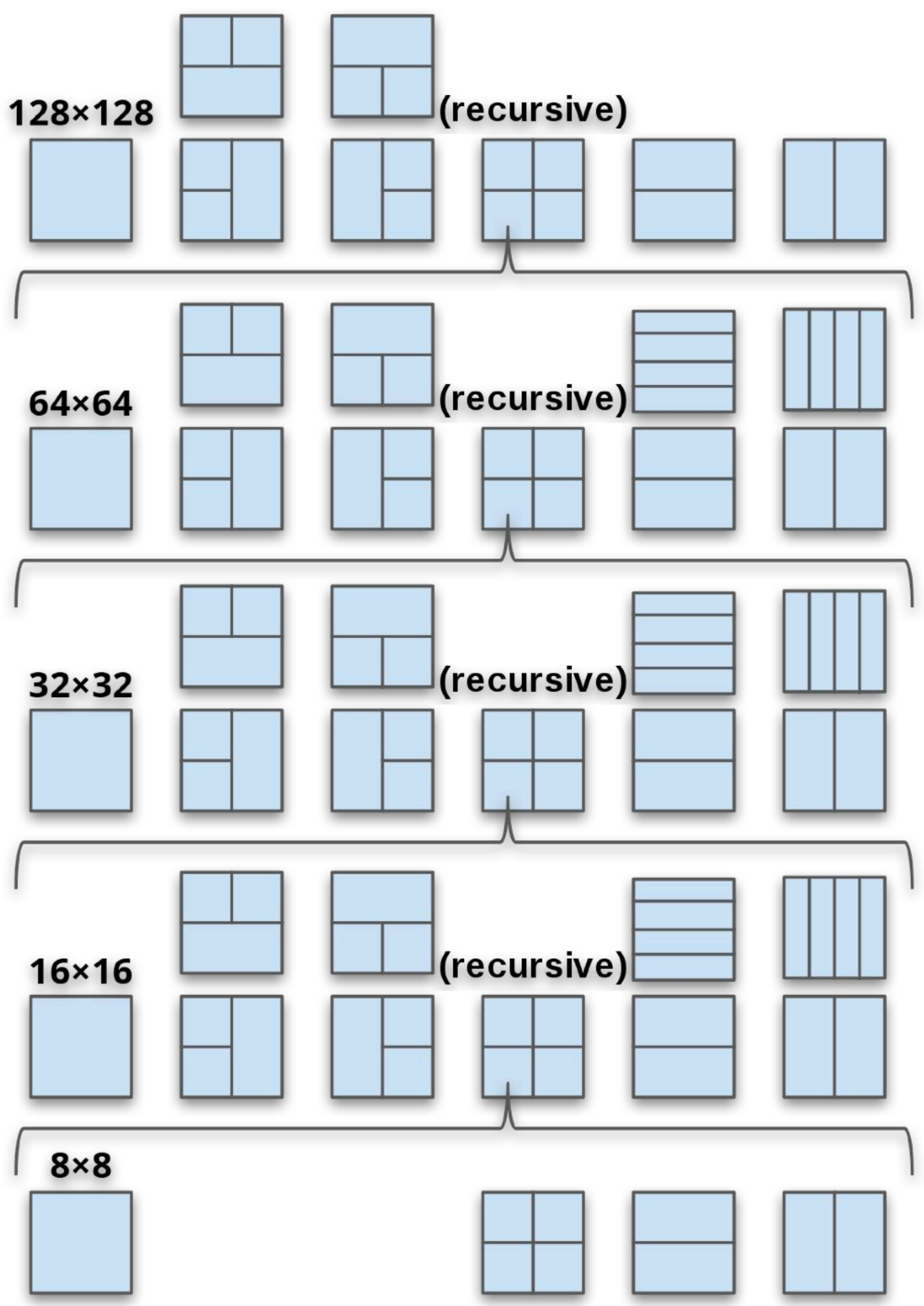
H.265 (HVEC)

- 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)
    - …

AV1

Main appeal may not be technical: royalty free codec, but many technical options for encoders

AV1 Superblock Partitionings



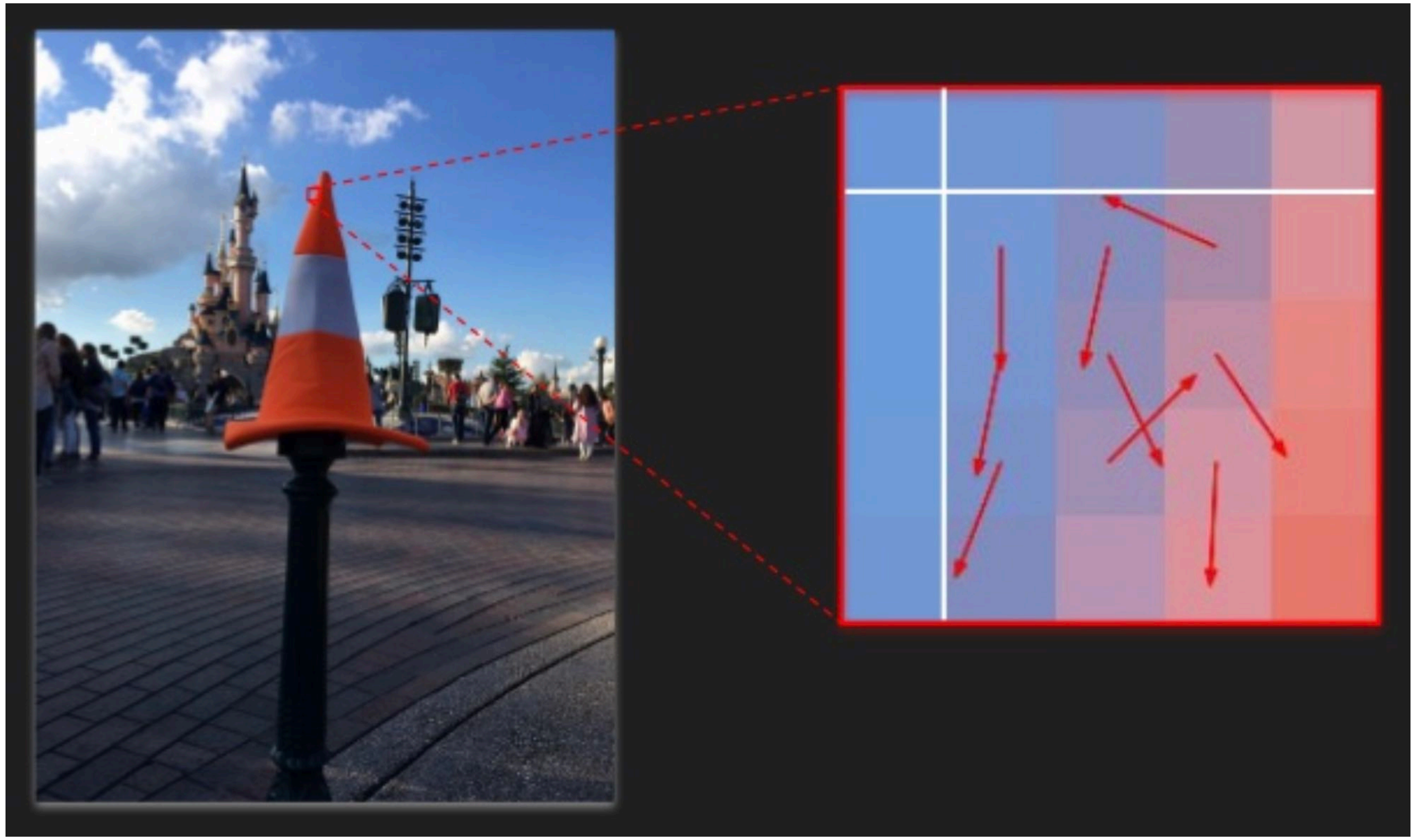
56 angles for intraframe block prediction!
(recall H.264 had nine!)

Global transforms to geometrically warp previous frames to new frames

Prediction of chroma channels from luma

Synthetic generation of film-grain texture so that high-frequency film grain does not need to be compressed…

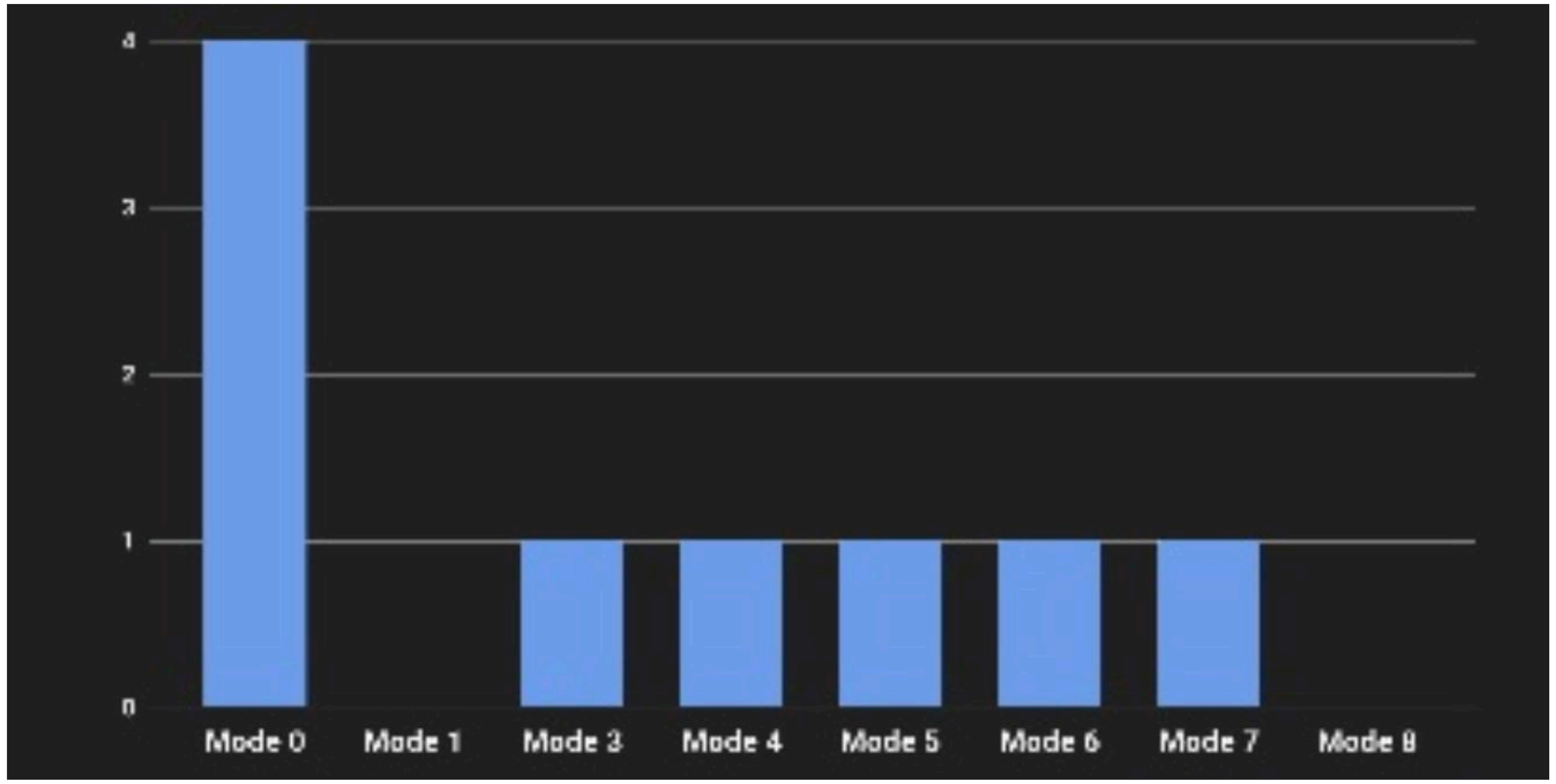
Example: searching for best intra angles



Compute image gradients in block

Bin gradients to find most likely to be useful angles.

Only try the most likely angles.



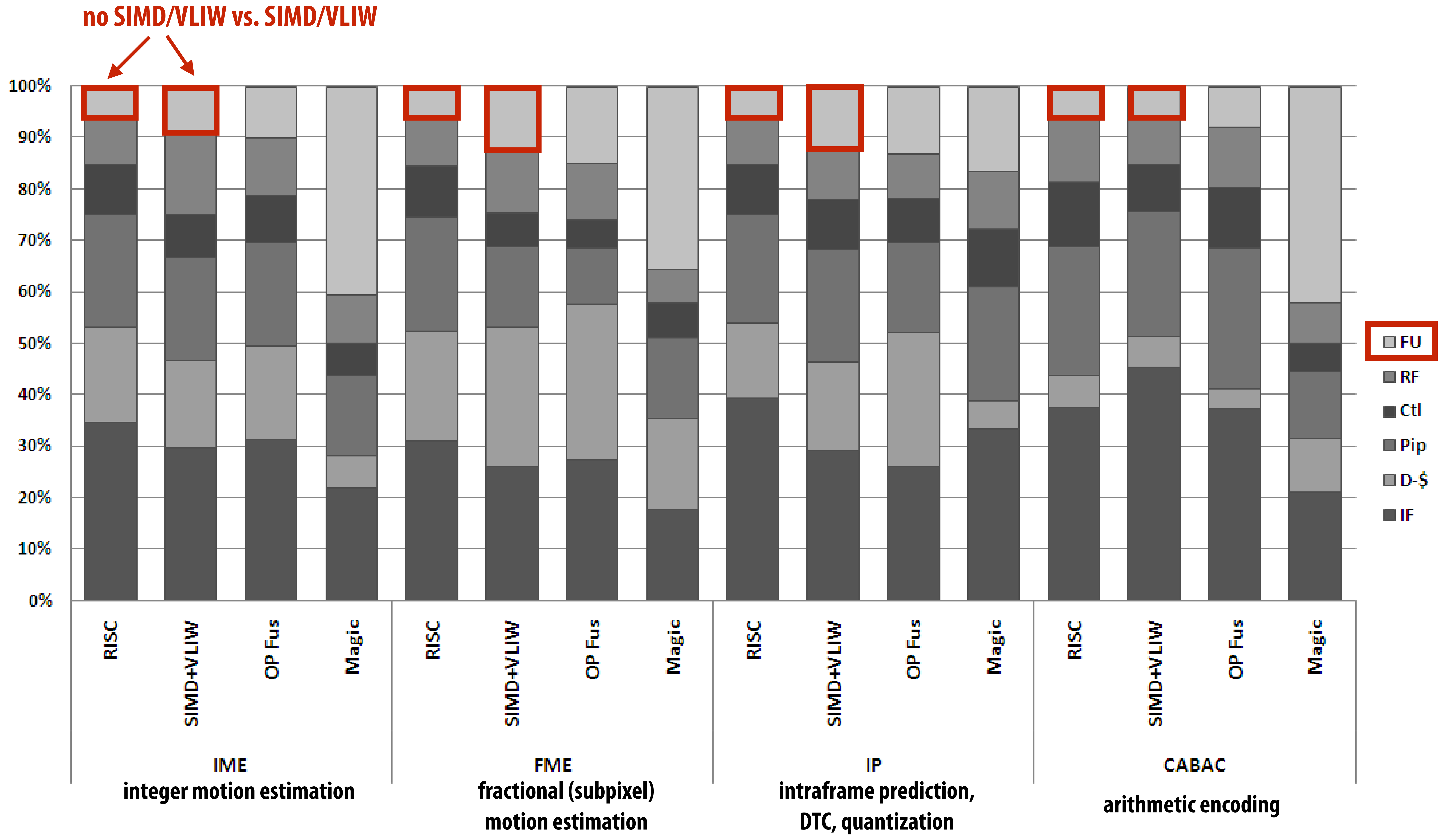
https://www.slideshare.net/luctrudeau/i-dont-care-if-you-have-360-intra-directional-predictors

High cost of software encoders

- **Statistic from Google:** [Ranganathan 2021]
  - About 8-10 CPU minutes to compress 150 frames of 2160p H.264 video
  - About 1 CPU hour for more expensive VP9 codec

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

[Hameed et al. ISCA 2010]

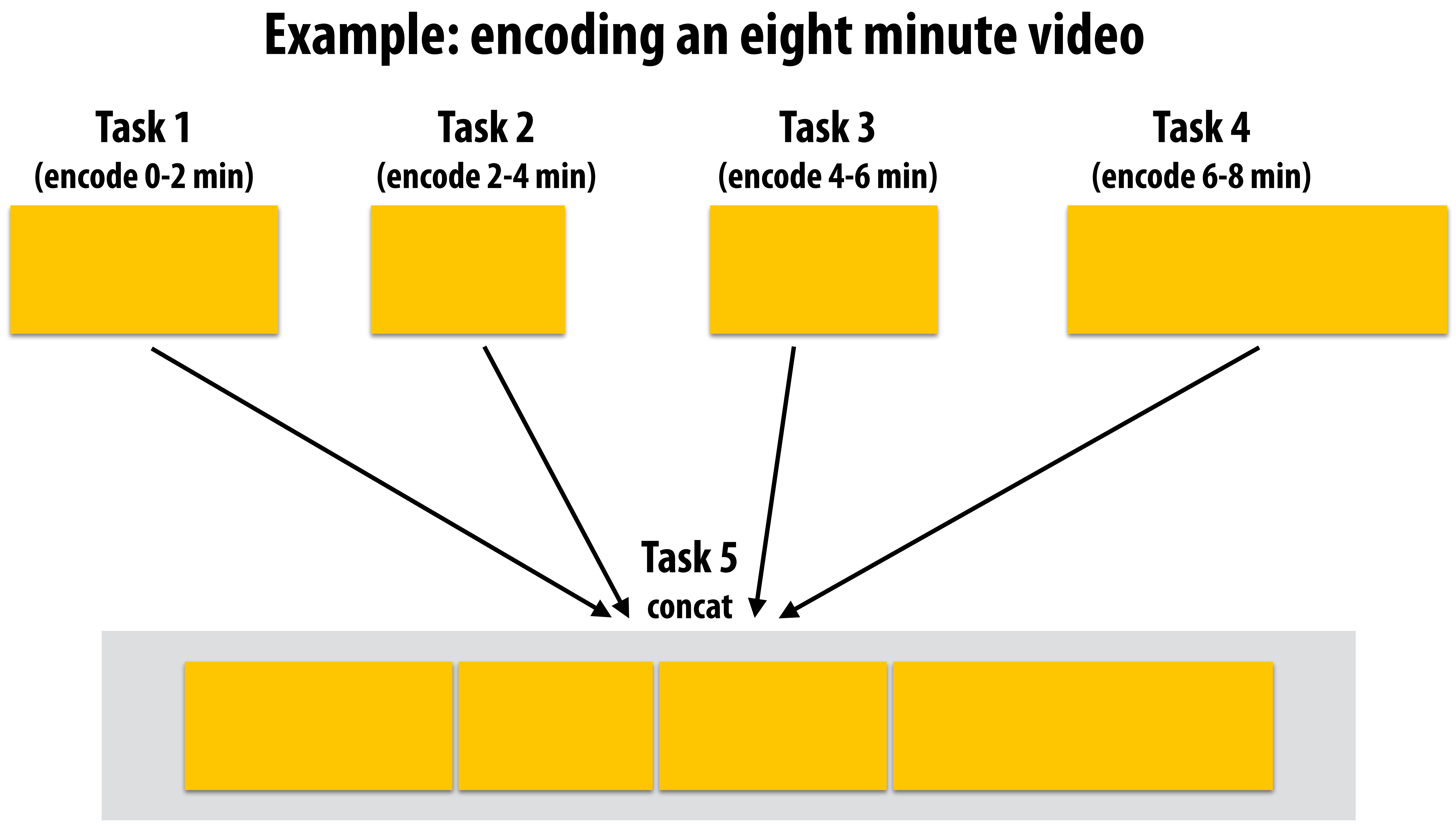


56% of total time	36% of total time	7% of total time	1% of total time (of baseline CPU imp)

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

Coarse-grained parallel video encoding

- Parallelized across segments (I-frame inserted at start of segment)
- Concatenate independently encoded bitstreams

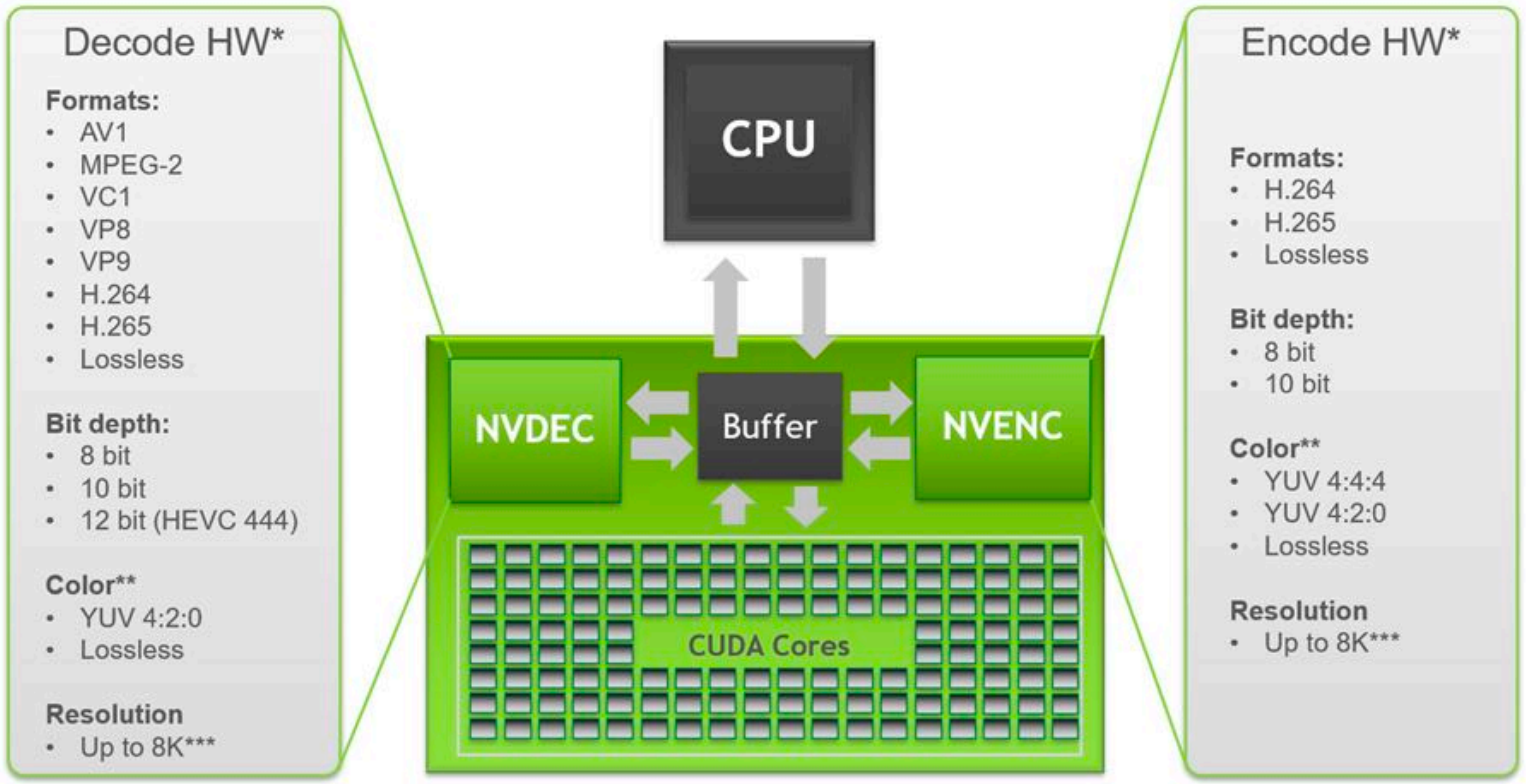


Smaller segments = more potential parallelism, worse video compression

ASIC acceleration of video encode/decode

NVIDIA GPUs have video encode/decode ASICs

- Example: GeForce NOW game streaming service
- Rendered images immediately compressed by GPU and bits streamed to remote player



- Another example: consumers at home streaming to Twitch
  - Do not want compression to take processing capability away from running the game itself.

Google's Video (Trans)coding Unit (VCU)

- ASIC hardware for decoding/encoding video in Google datacenter for Youtube/Youtube Live
- Consider load:
  - 500 hours of video uploaded to Youtube per minute (2019)
  - Must support streaming to consumers with many different devices and networks (must generate encoded versions assets at many resolutions and using different codecs)

System	Throughput [Mpix/s]		Perf/TCO[8]	
	H.264	VP9	H.264	VP9
Skylake	714	154	1.0x	1.0x
4xNvidia T4	2, 484	—	1.5x	—
8xVCU	5, 973	6, 122	4.4x	20.8x
20xVCU	14, 932	15, 306	7.0x	33.3x

[Ranganathan 2021]

Learning Compression Schemes

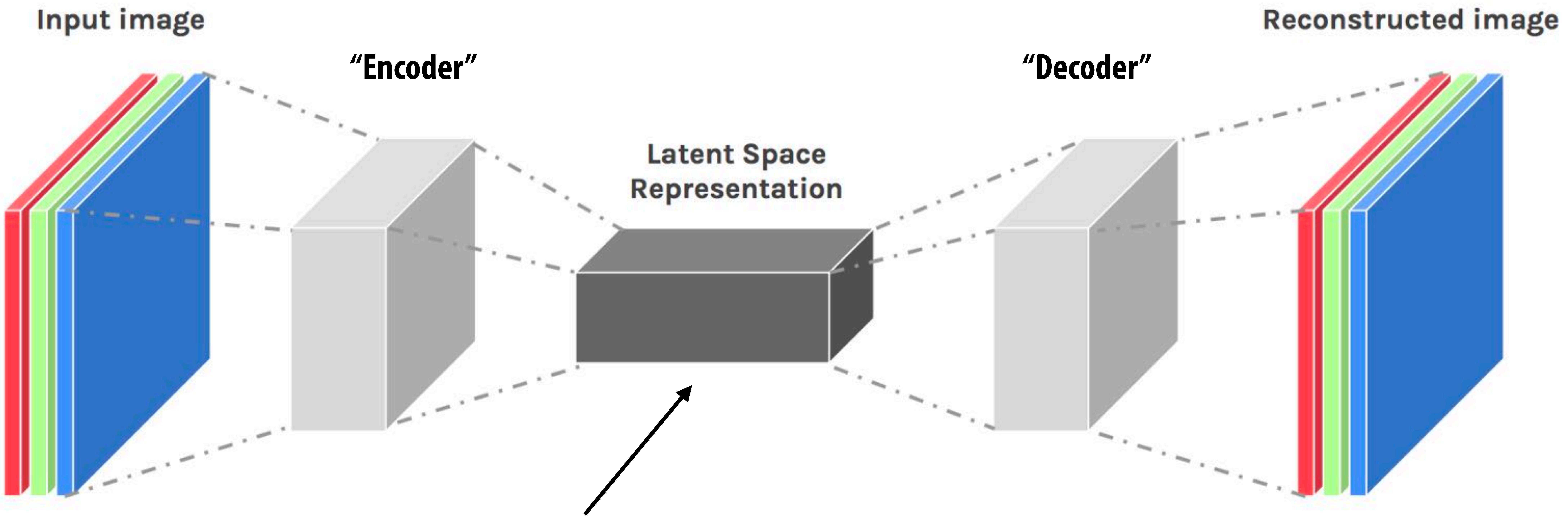
Learned compression schemes

- **JPG image compression and H.264/265/AV1 video compression are "lossy" compression techniques that discard information is that is present in the visual signal, but less likely to be noticed by the human eye**
  - **Key principle: "Lossy, but still looks good enough to humans!"**

- **Compression schemes described in this lecture involved manual choice / engineering of good representations (features)**
  - **Frequency domain representation, YUV representation, disregarding color information, flow vectors, etc.**

- **Increasing interest in *learning* good representations for a specific class of images/videos, or for a *specific task* to perform on images/videos**

DNN autoencoder

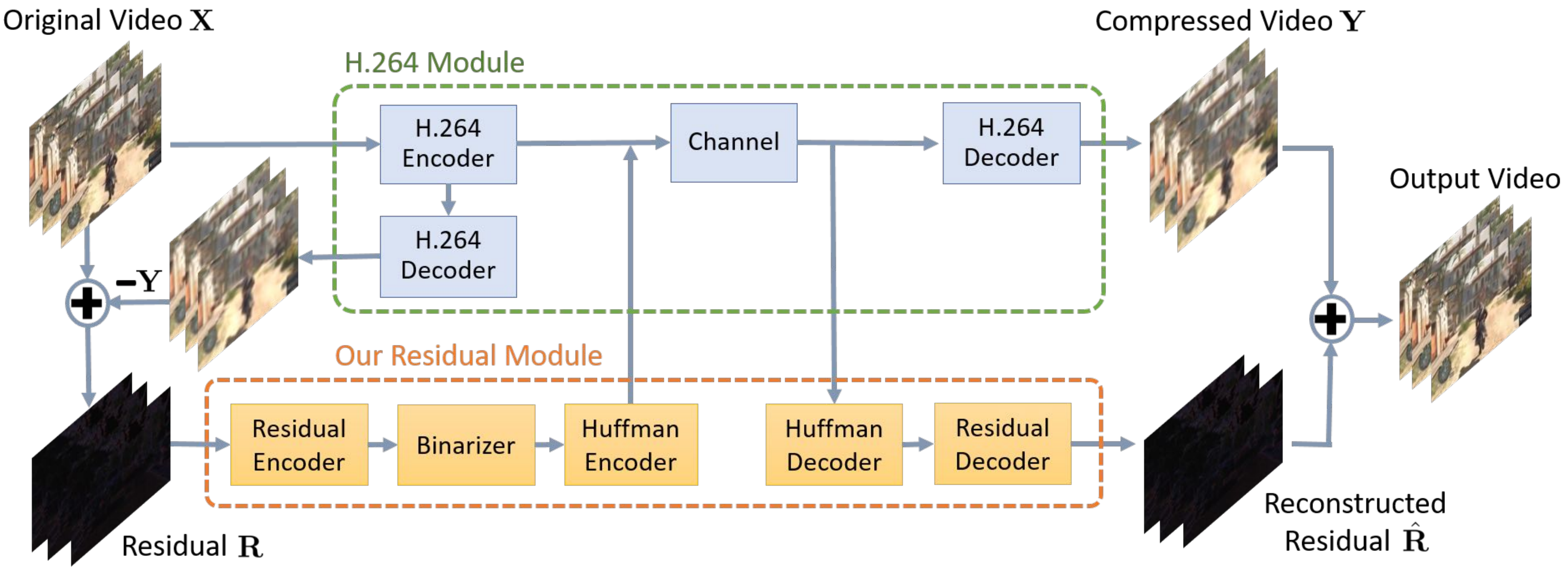


If this latent representation is compact, then it is a compressed representation of the input image

https://medium.com/@birla.deepak26/autoencoders-series-daad78df9350

Learned compression schemes

Many recent DNN-based approaches to compressing video learn to *compress the residual*



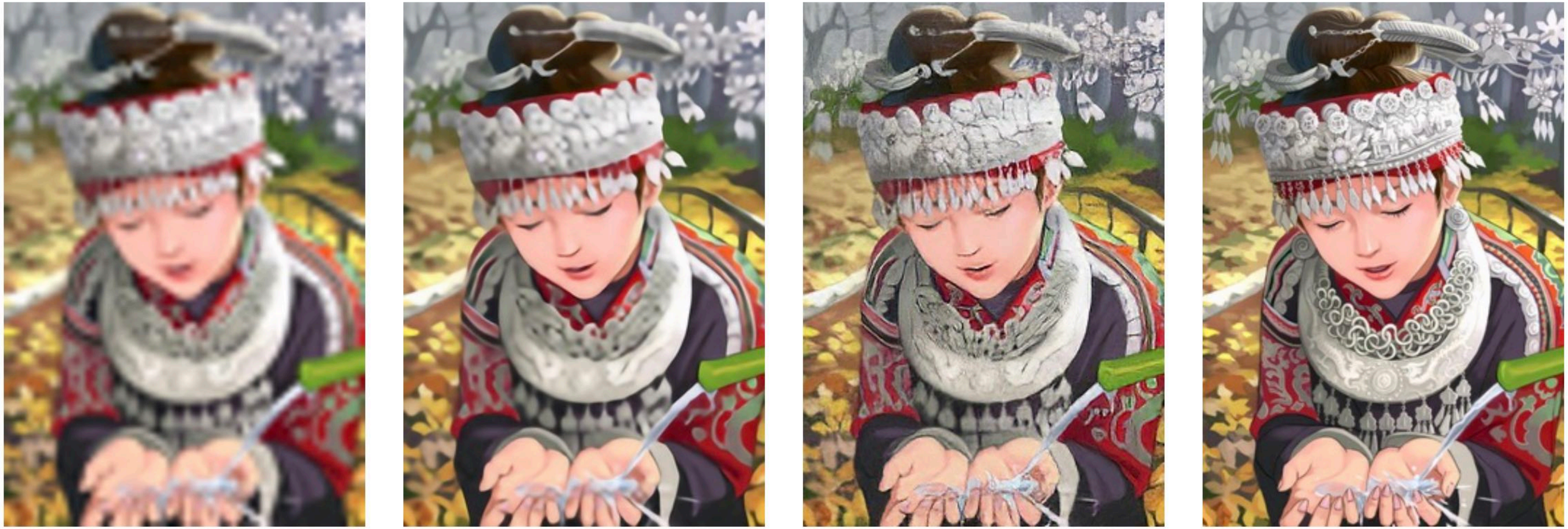
[Tsai et al. 2018]

Use standard video compression at low quality, then use an autoencoder to compress the residual.
(Learn to compress the residual)

Super-resolution-based reconstruction

- **Single image superresolution: given a low-resolution image, predict the corresponding high resolution image**

bicubic (21.59dB/0.6423) | SRResNet (23.53dB/0.7832) | SRGAN (21.15dB/0.6868) | original



[SRGAN, Ledig et al. CVPR 2017]

Super-resolution-based reconstruction

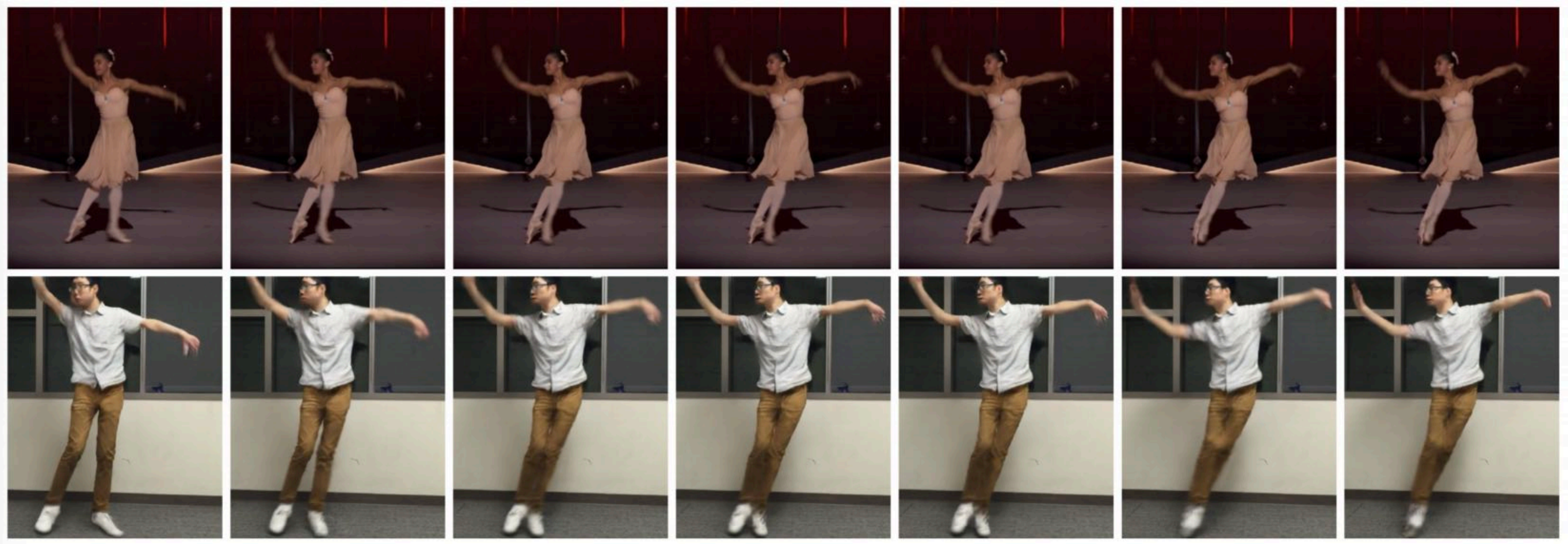
- Encode low-resolution video using standard video compression techniques
- Also transfer (as part of the video stream) a video-specific super-resolution DNN to upsample the low resolution video to high res video.
  - Assumption: training costs are amortized over many video downloads

(a) Original (1080p) (b) Content-aware DNN (c) Content-agnostic DNN (d) 240p

[Yeo et al. OSDI 2018]

Person specific compression

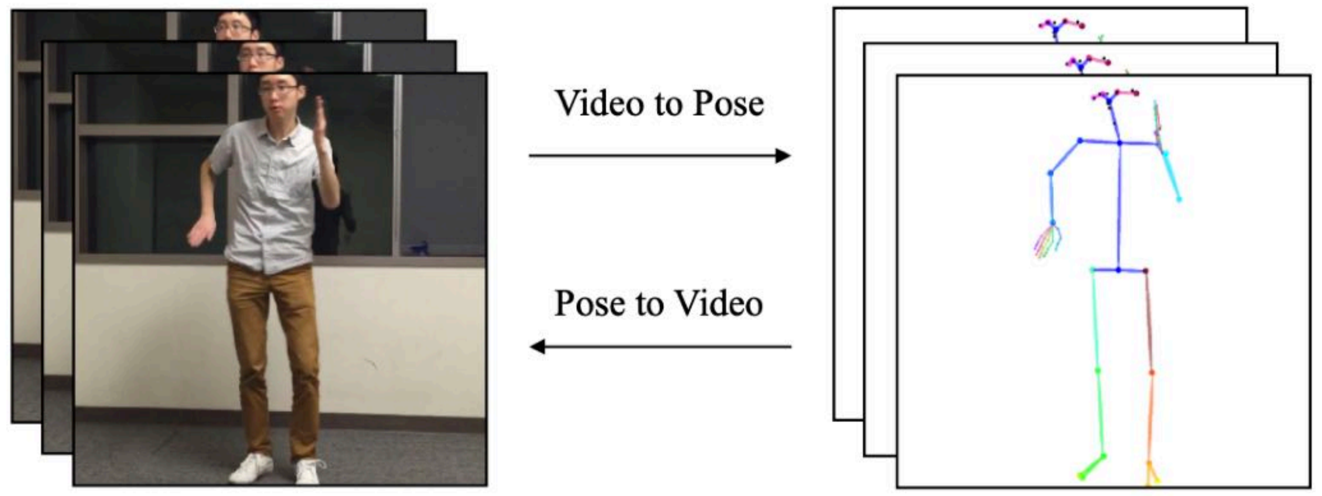
Input: video of professional ballerina performing a motion



Output: video of graduate student performing the same motion

Input video Frame → Pose Detector DNN → 2D Pose → Pose-2-image DNN → Output video frame

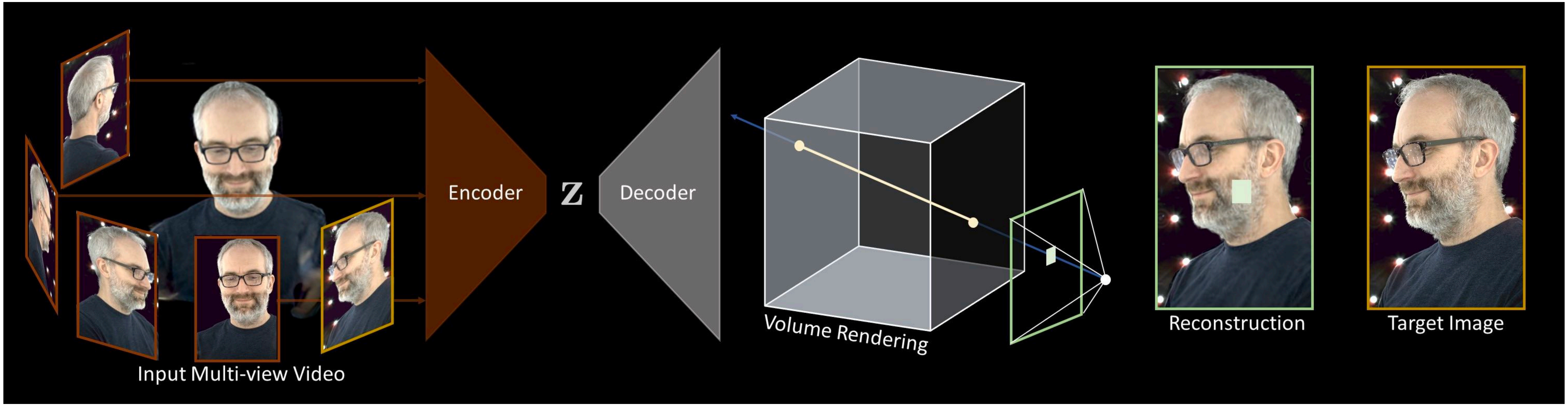
Encode video as just a set of 14 pose joints.



[Chan et al. 2019]

Neural volumes

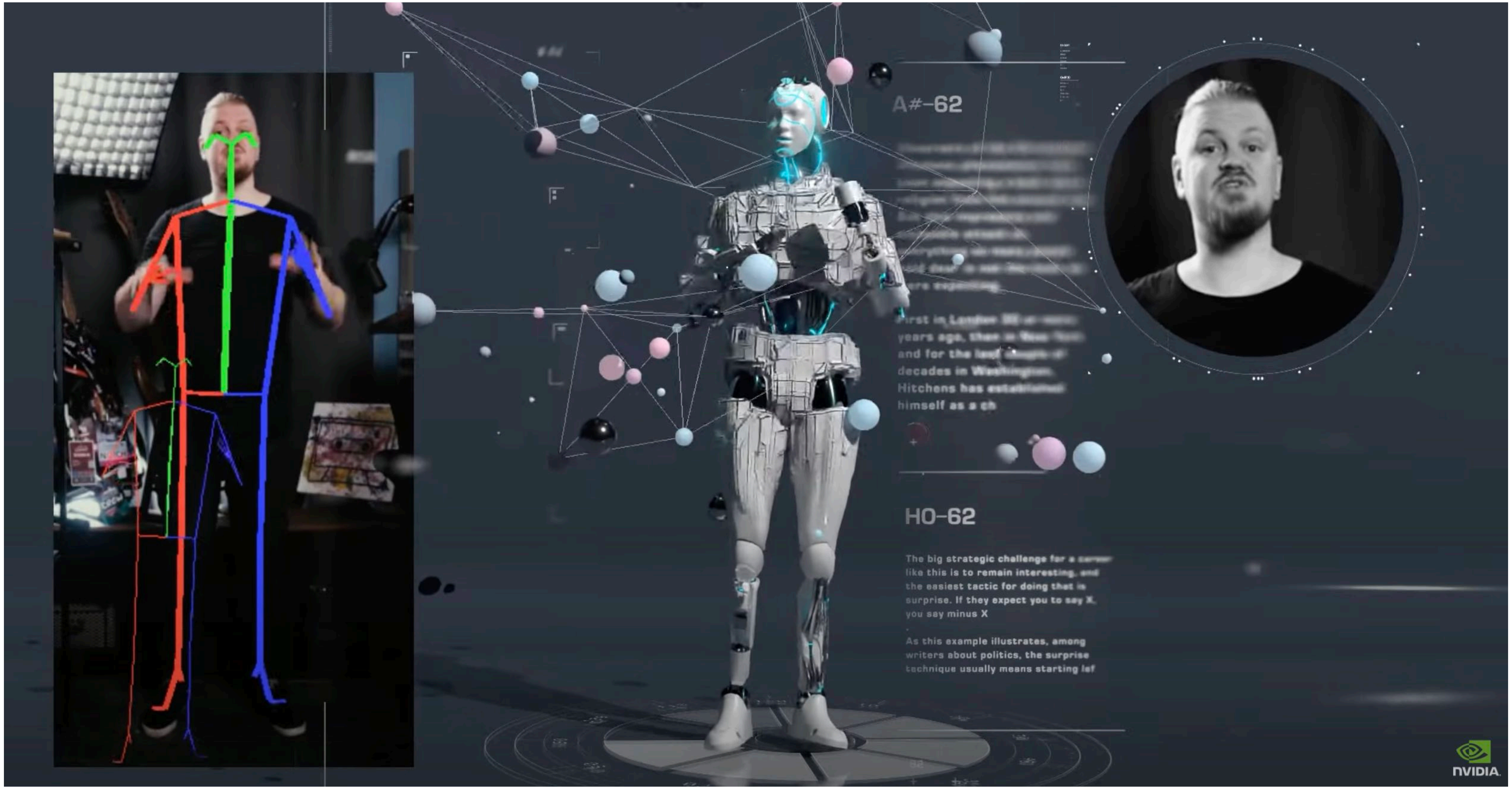
- Learn to encode multiple views of a person into a latency code (z) that is decoded into a volume than can be rendered with conventional graphics techniques *from any viewpoint*



- Motivated by VR applications

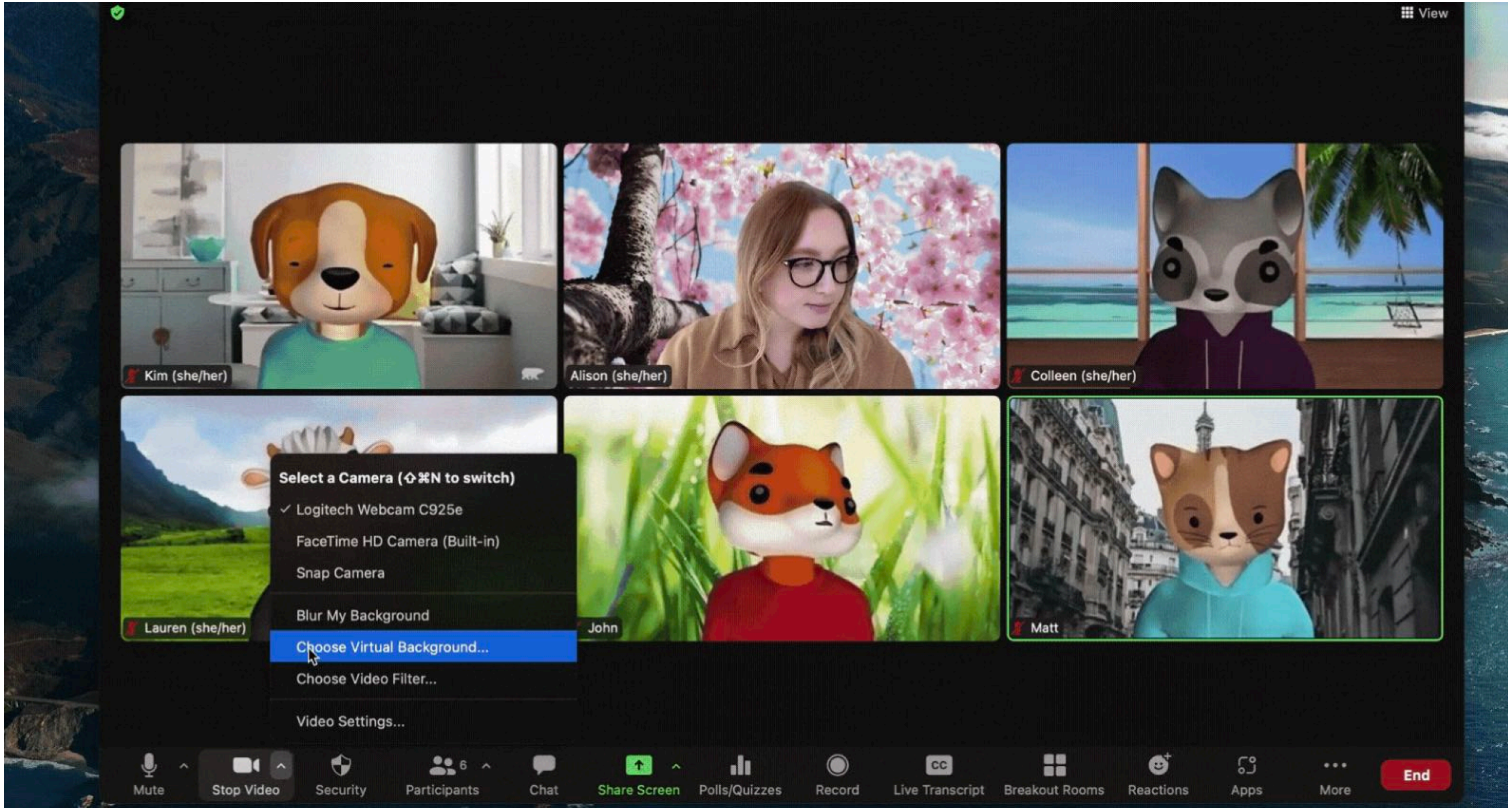
NVIDIA Maxine

GPU-accelerated video processing for video conferencing applications



Examples: avatar control, video superresolution, advanced background segmentation

Zoom avatars / Snapcam Lenses



Where is the line between transmission of what happened and "making something up"?
More on this next class.

Summary

- JPG image compression and H.264/265/AV1 video compression are "lossy" compression techniques that discard information is that is present in the visual signal, but less likely to be noticed by the human eye
  - Key principle: "Lossy, but still looks good enough to humans!"
- Key idea of video encoding is "searching" for a compact encoding of the visual signal in a large space of possibilities
  - Video encoder ASIC used to accelerate this search
- Growing interest in learning these encodings, but hard to beat well-engineered features
  - But promising if learned features are specialized to video stream contents
  - Or to specific tasks (remember, increasing amount of video is not meant to be consumed by human eyes)