Lecture 20:

Video Compression

Visual Computing Systems
CMU 15-869, Fall 2013

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 per fixed bit rate
- Costs: decoding complexity, substantially higher encoding cost
  - Trade more compute for less bandwidth/storage

Hardware implementations

- Support for encode/decode provided via fixed function hardware on modern mobile devices

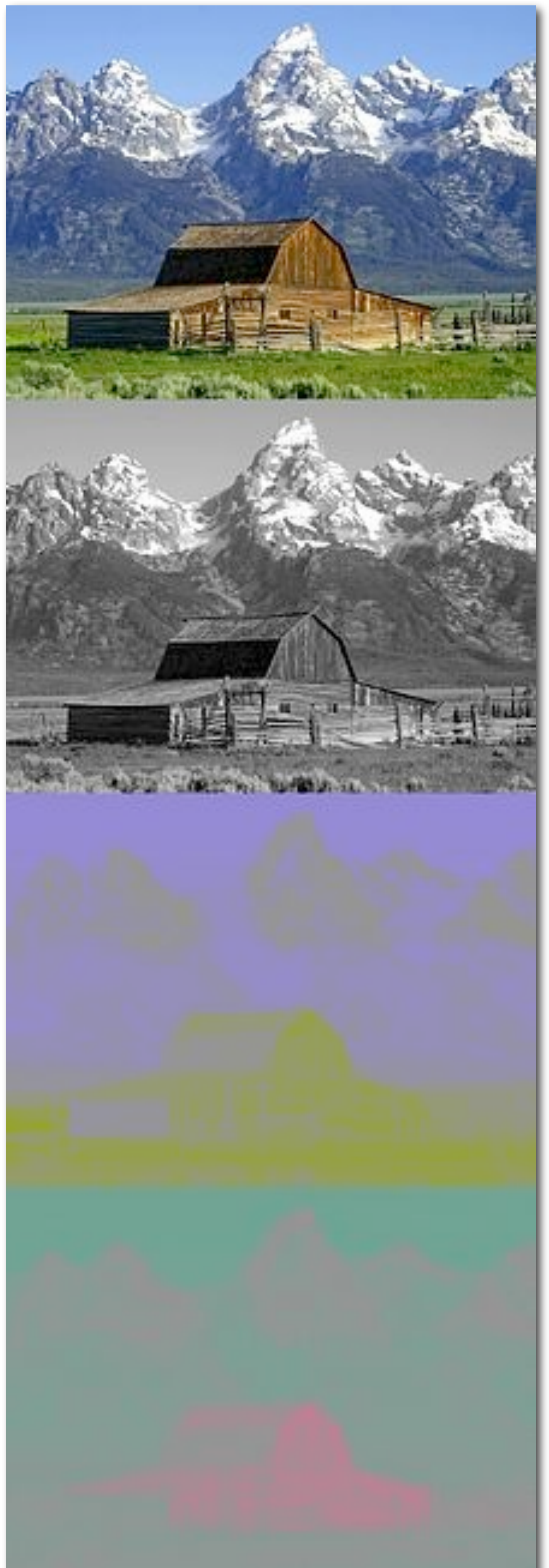
- Hardware encoding/decoding support in modern Intel CPUs since Sandy Bridge (Intel "Quick Sync")

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

Video container format versus 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

Review: Y'CbCr 4:2:0



Y'

Cb

Cr

Y' = perceived brightness ("luma")

Cb = blue-yellow deviation from gray

Cr = red-cyan deviation from gray

4:2:0 representation (subsampled chroma):

- Store Y' at full resolution
- Store Cb, Cr at half vertical and horizontal resolution (1/4 as many chroma samples as luminance samples)

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

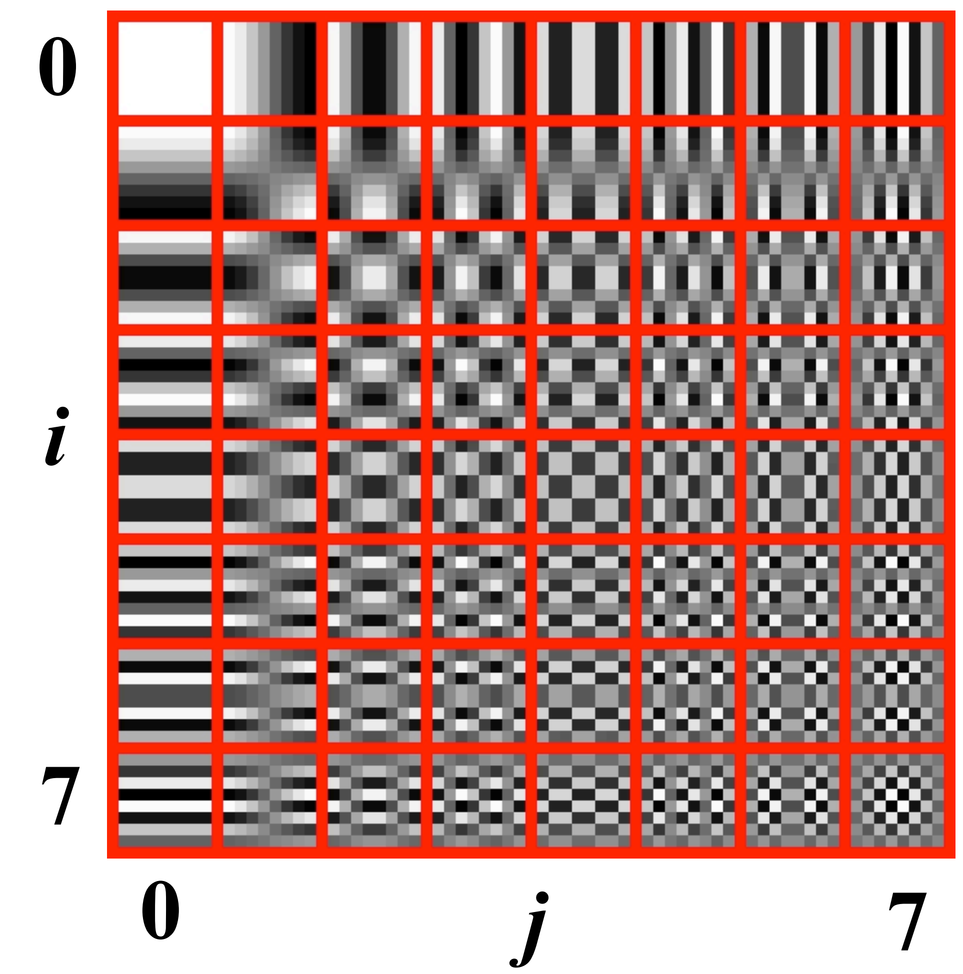
Image credit: Wikipedia

Review: image transform coding via DCT

(JPEG compression segment of camera pipeline lecture)

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

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



Note: only showing coefficients for one channel (e.g., Y') here. Each channel is transformed independently.

Credit: Wikipedia, Pat Hanrahan

Review: quantization and entropy compression

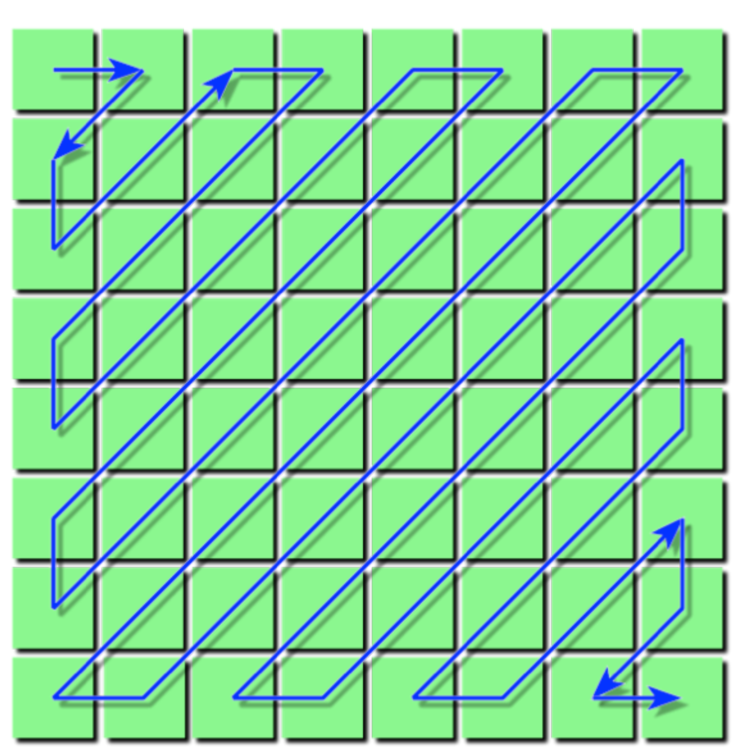
$$
\begin{bmatrix}
-415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\
4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\
-47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\
-49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\
12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\
-8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\
-1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\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

DCT / **Quantization Matrix**

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

Quantized DCT

Quantization loses information (lossy compression!)



Coefficient reordering

RLE compression of zeros

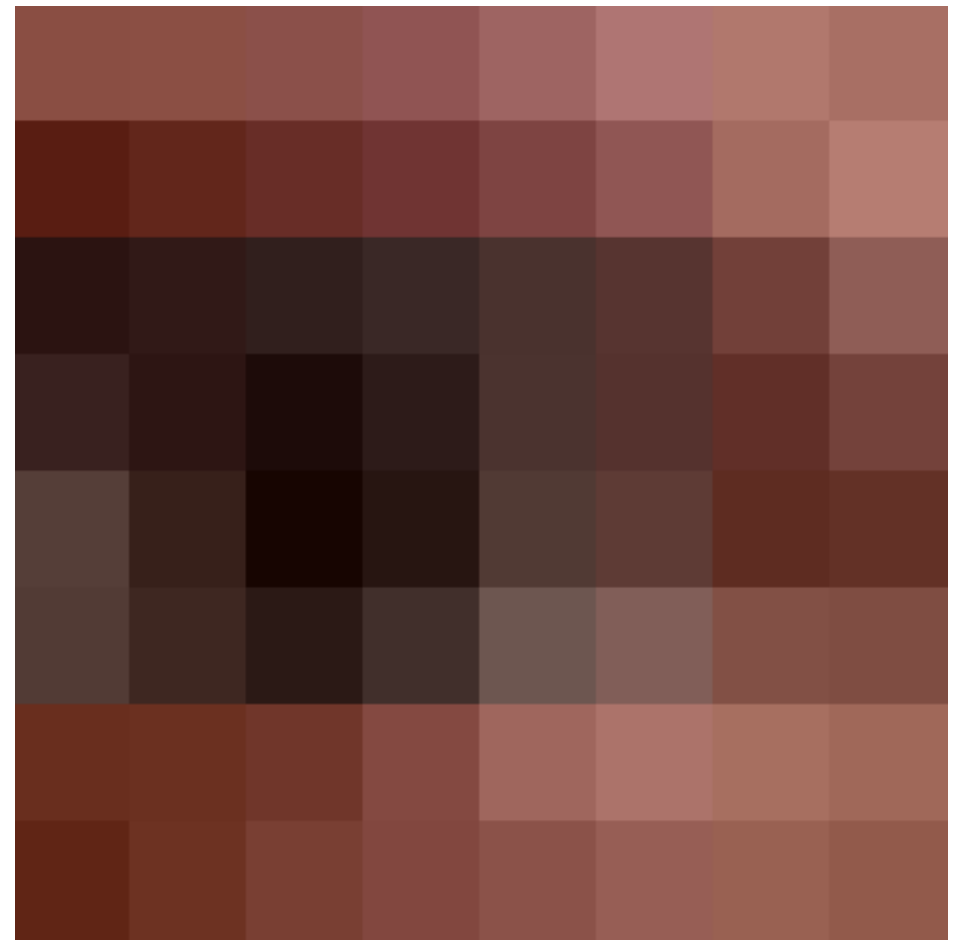
Entropy compression of non-zeros

Lossless compression!

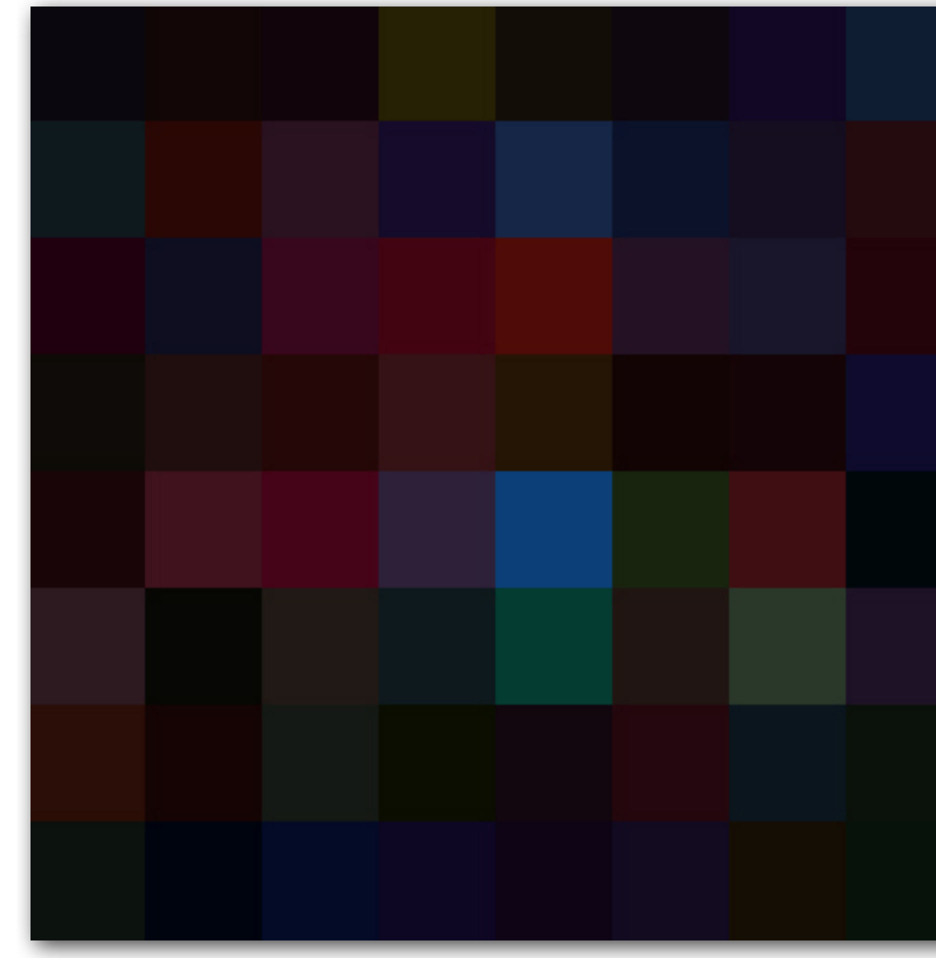
Compressed bits

Residual: difference between compressed and original image

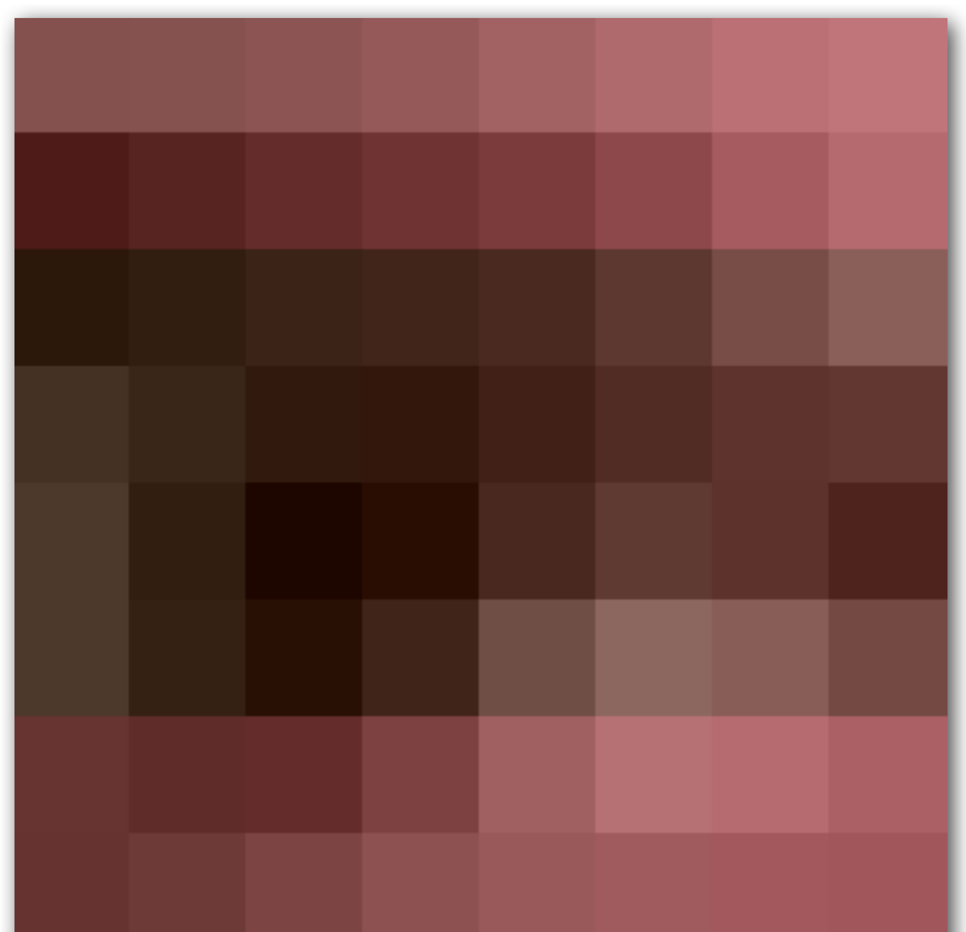
Original pixels



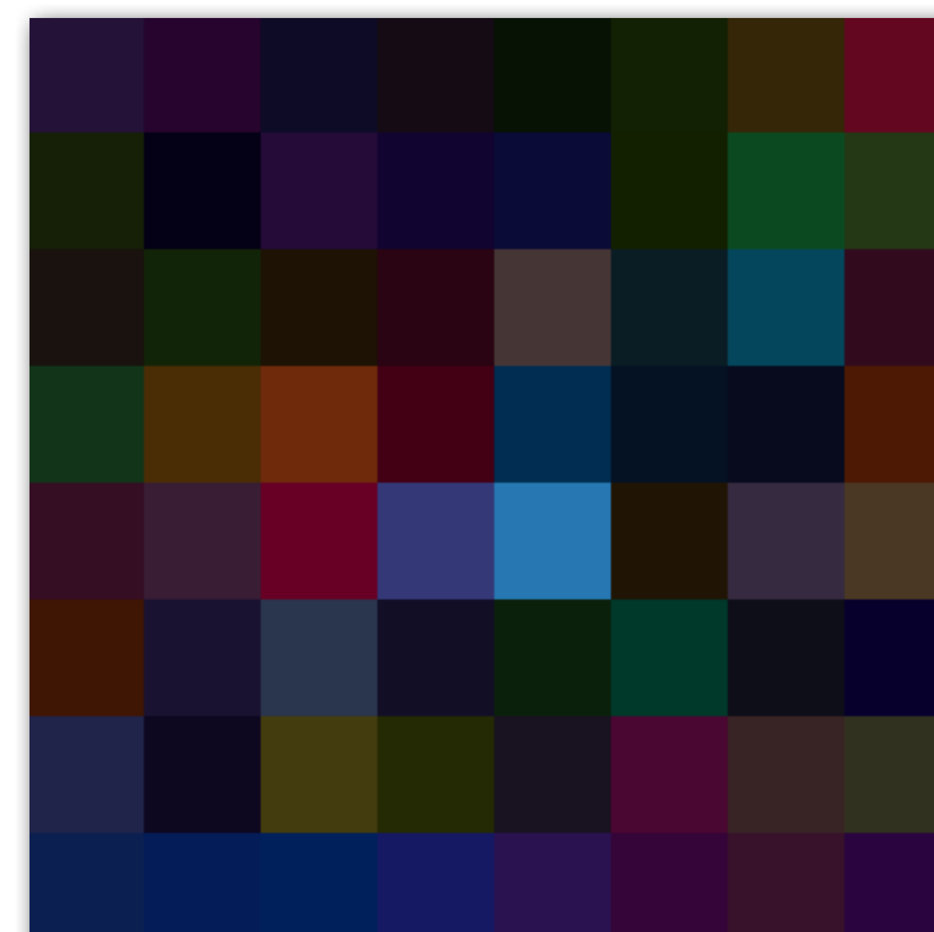
Compressed pixels
(JPEG quality level 6)



Residual
(amplified for visualization)



Compressed pixels
(JPEG quality level 2)

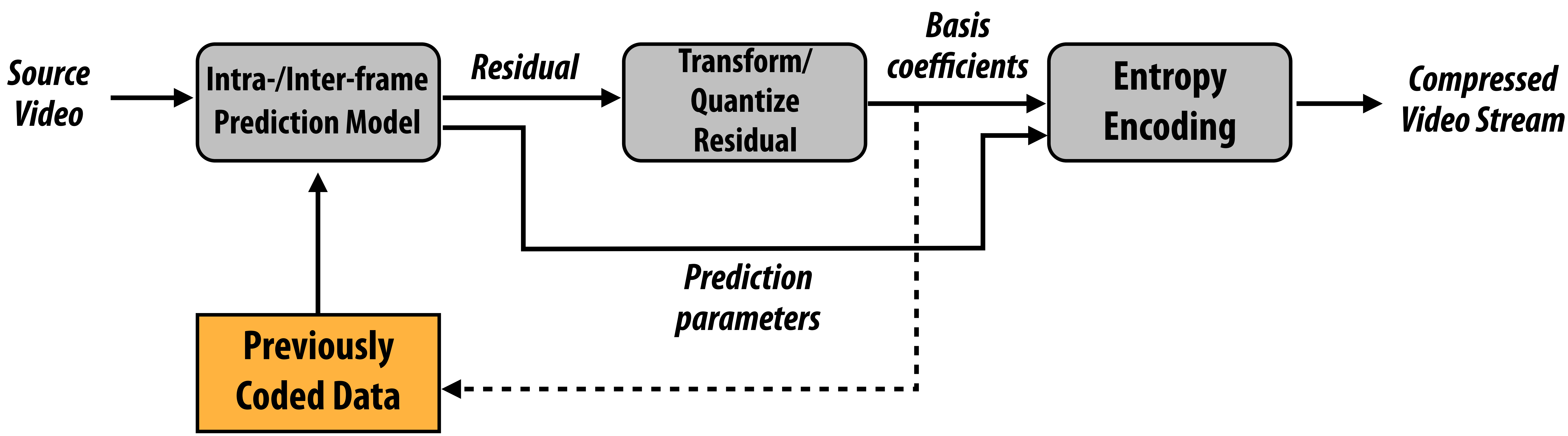


Residual
(amplified for visualization)

Video compression main ideas

- Compression is about exploiting redundancy in signal
  - Intra-frame redundancy: pixels in neighboring regions of a frame are good indicator of other pixels in the frame
  - Inter-frame redundancy: pixels from nearby frames in time are a good predictor for current frame pixels

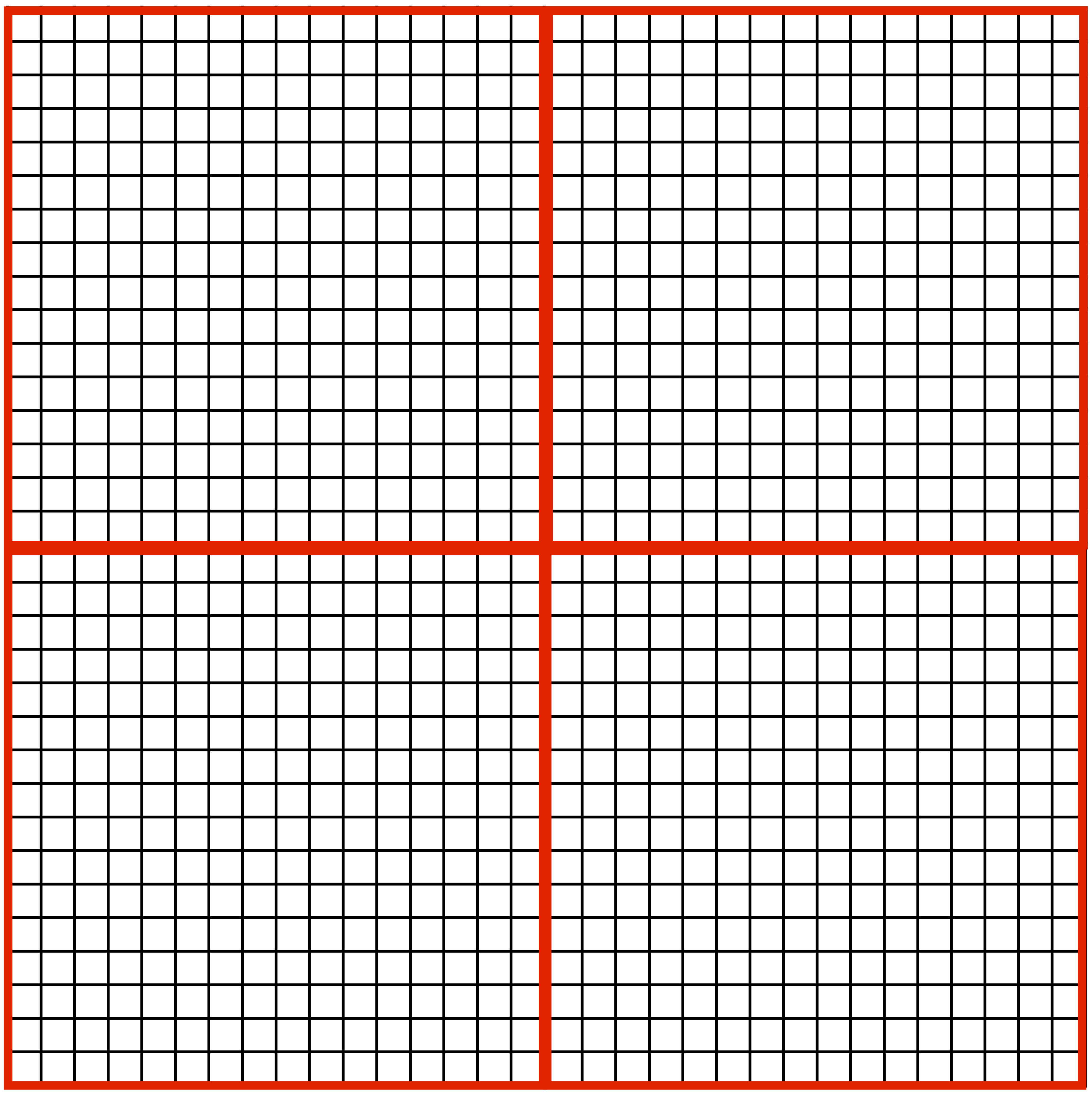
H.264/AVC video compression overview



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

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

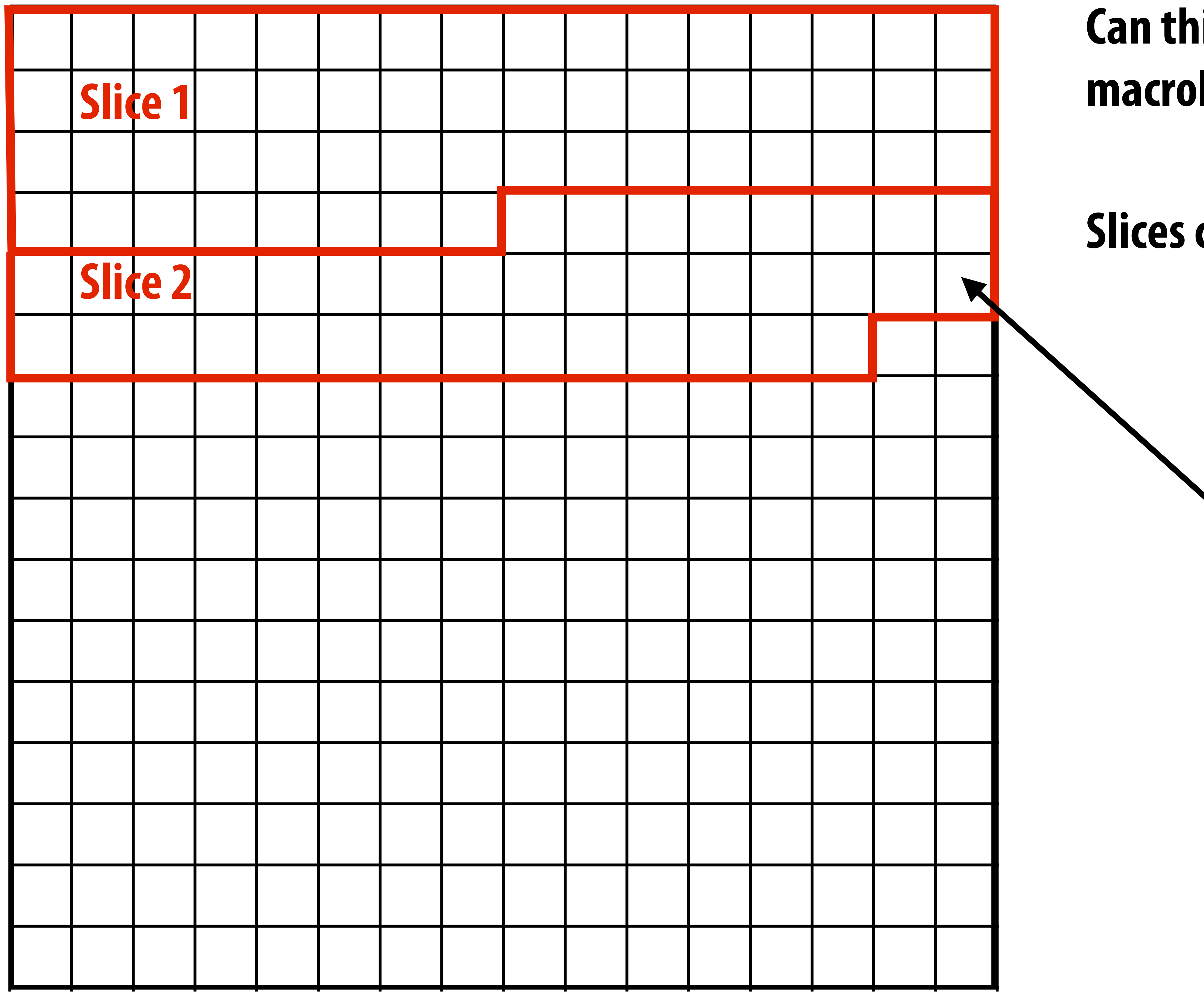
16 x 16 macroblocks



Frame 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 organized 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.

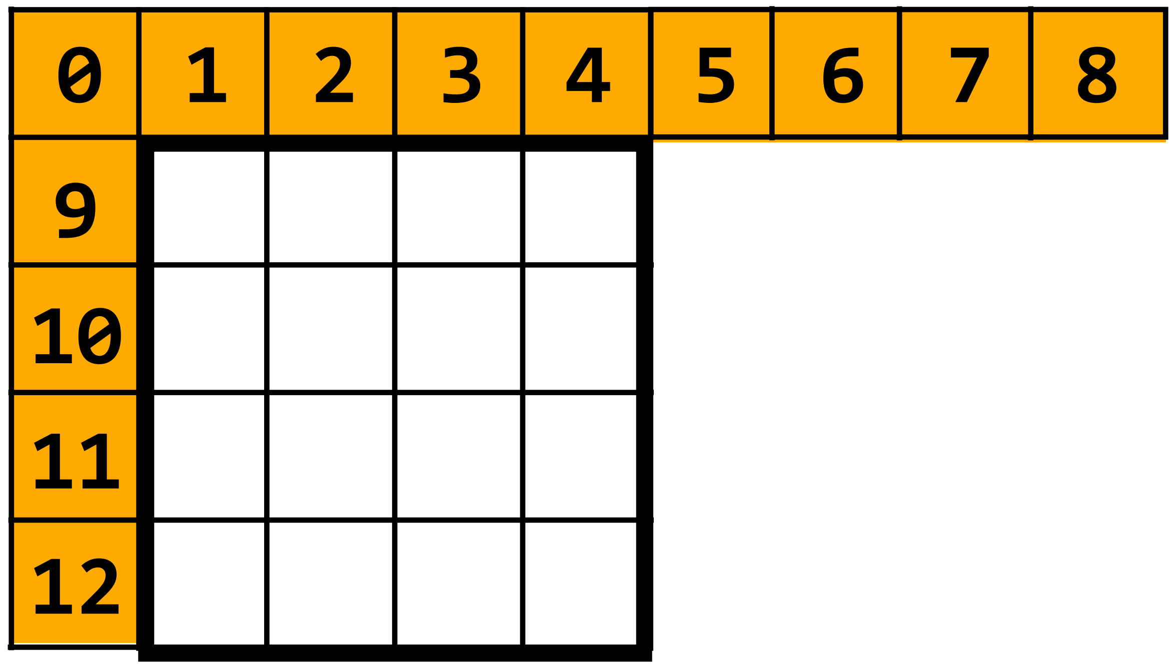
Decoding via prediction + correction

- During decode, samples in a macroblock are generated by:
  1. Making a prediction based on already decoded samples in macroblocks from the same frame (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: macroblock samples predicted from samples in previous macroblocks in the same slice of the current frame
  - P-macroblock: macroblock samples can be predicted from samples from one other frame (one prediction per macroblock)
  - B-macroblock: macroblock samples can be predicted by a weighted combination of multiple predictions from samples from other frames

Intra-frame prediction (I-macroblock)

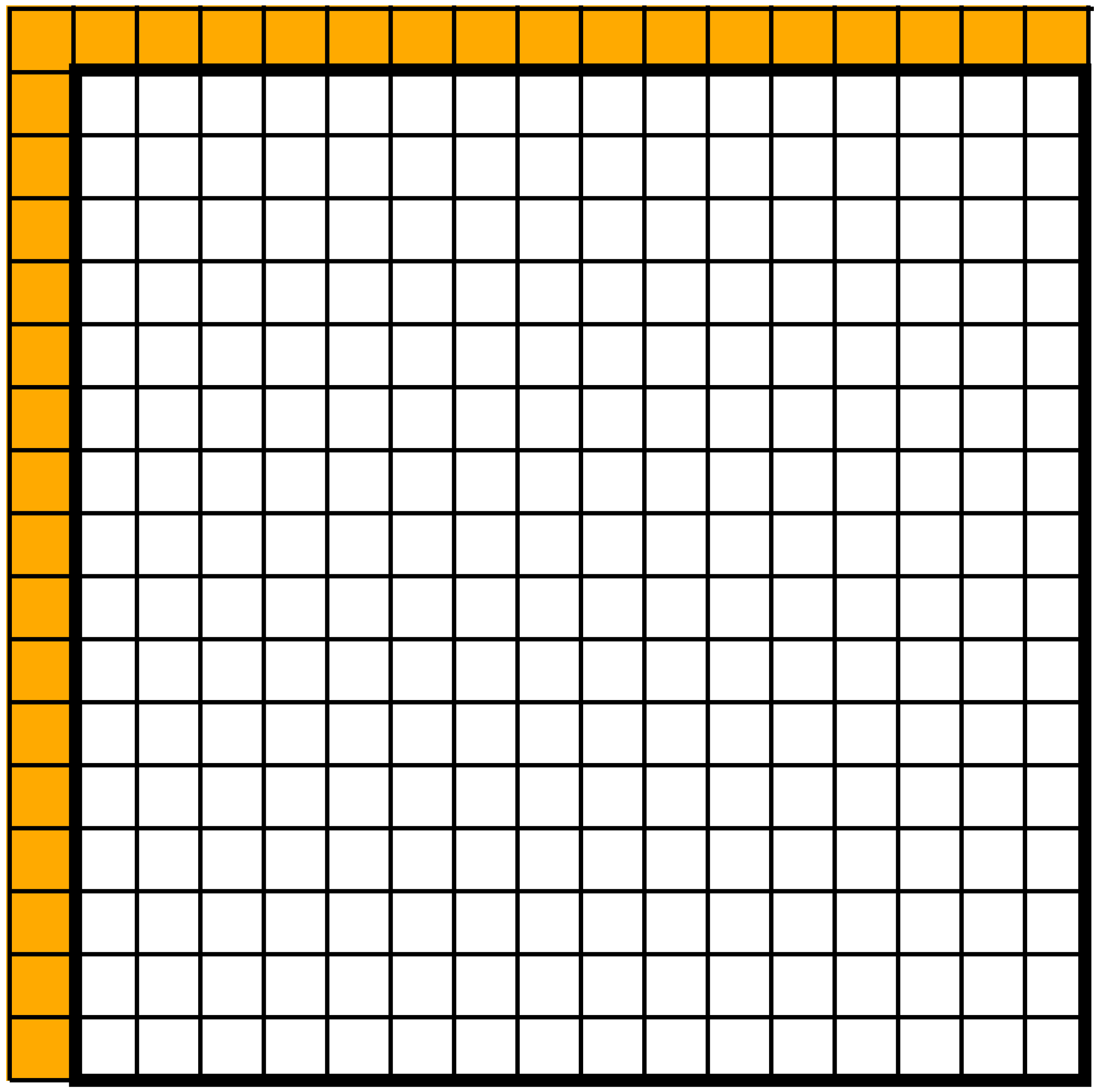
- Prediction of sample values is performed in spatial domain, not transform domain
  - Predicting pixel values, not basis coefficients
- Three modes for predicting luma (Y):
  - Intra_4x4 mode: predict 4x4 block of samples from adjacent row/col of pixels
  - Intra_16x16 mode: predict entire 16x16 block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



Intra_4X4

Yellow pixels: already reconstructed (values known)

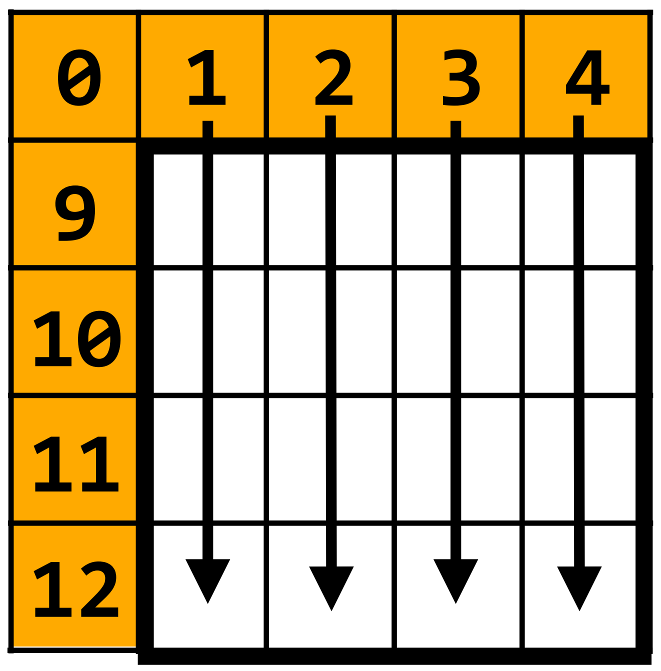
White pixels: 4x4 block to be reconstructed



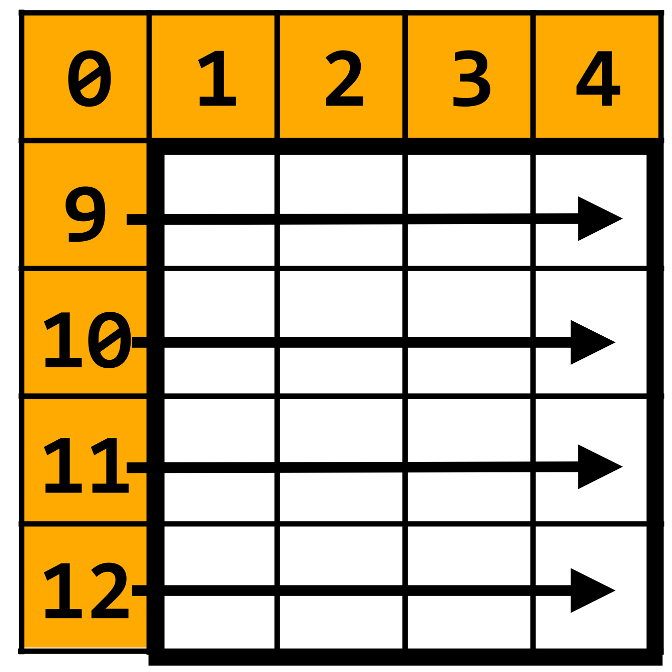
Intra_16x16

Intra_4x4 prediction modes

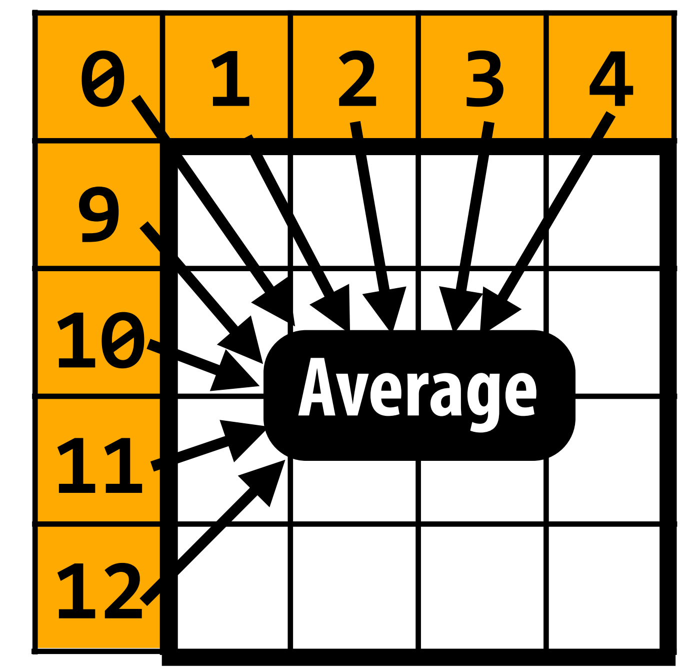
- Nine prediction modes (5 shown below)
  - Other modes: vert-right, horiz-down, vertical-left, horiz-up



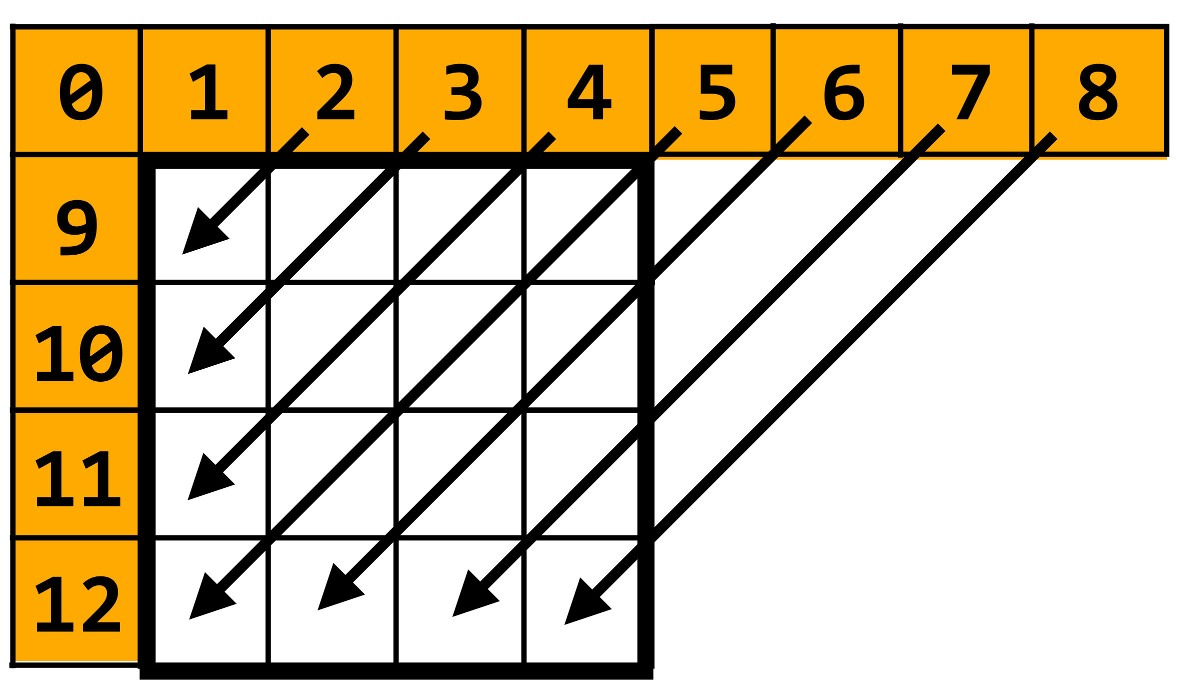
Mode 0: vertical
(4x4 block is copy of above row of pixels)



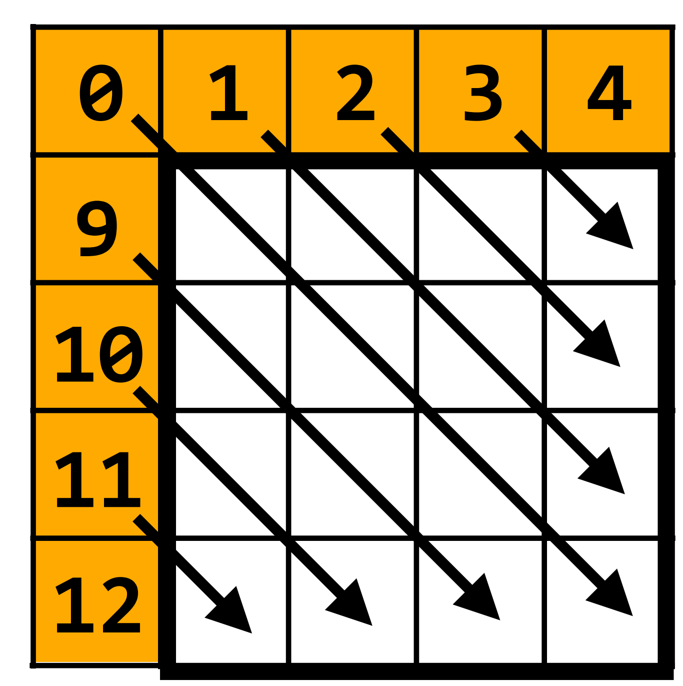
Mode 1: horizontal
(4x4 block is copy of left col of pixels)



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



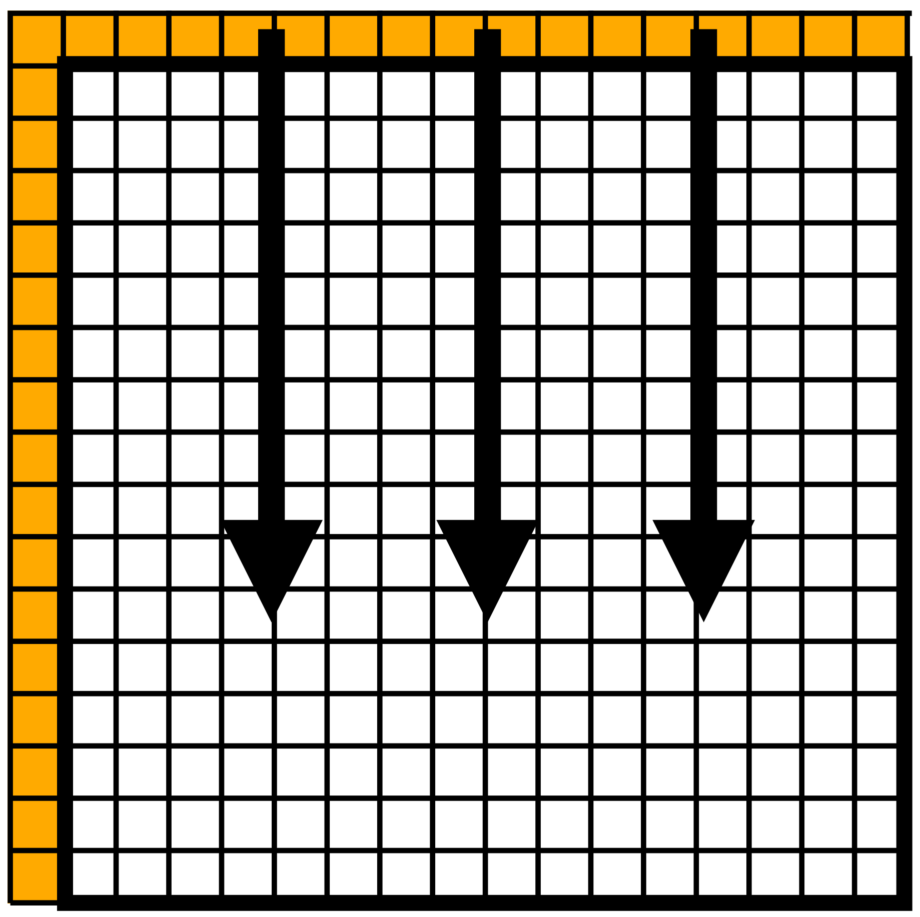
Mode 3: diagonal down-left



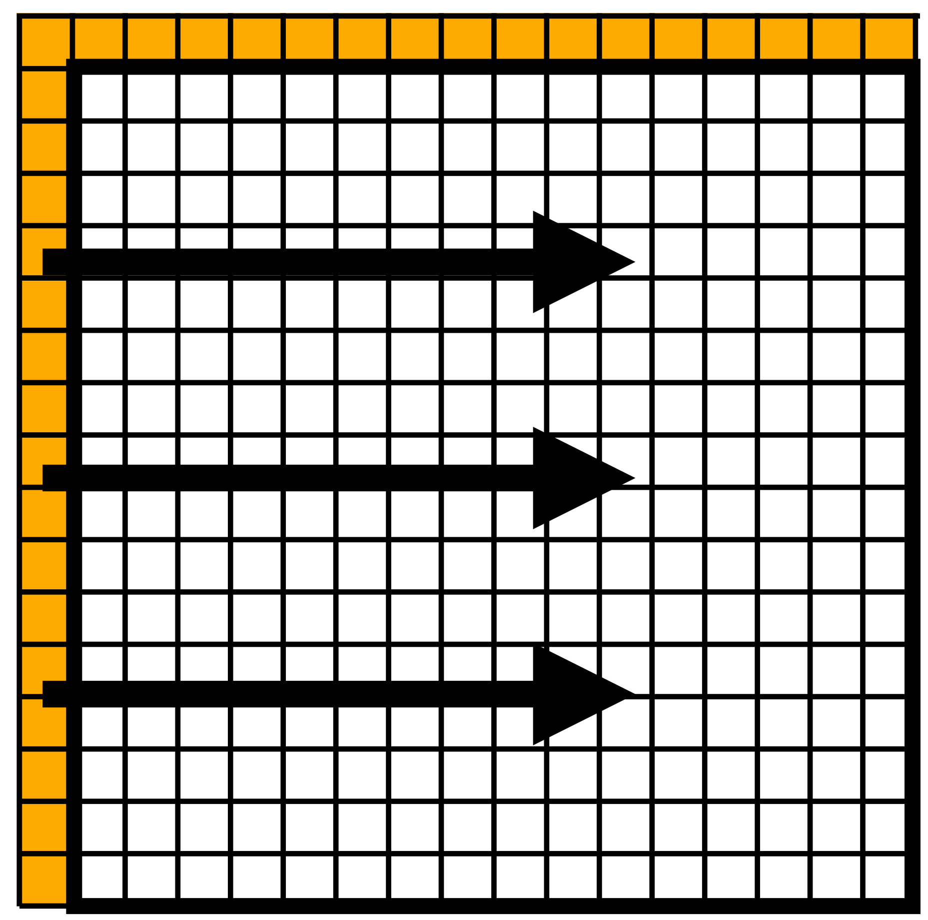
Mode 4: diagonal down-right

Intra_16x16 prediction modes

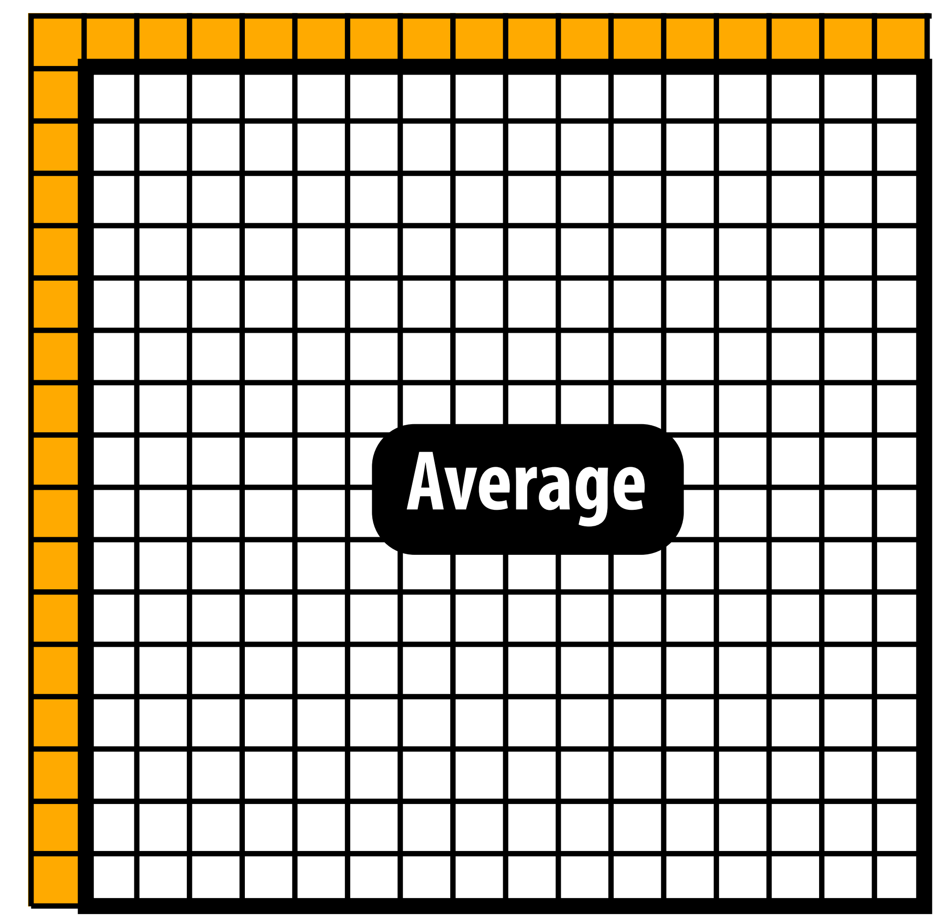
- **4 prediction modes: vertical, horizontal, DC, plane**



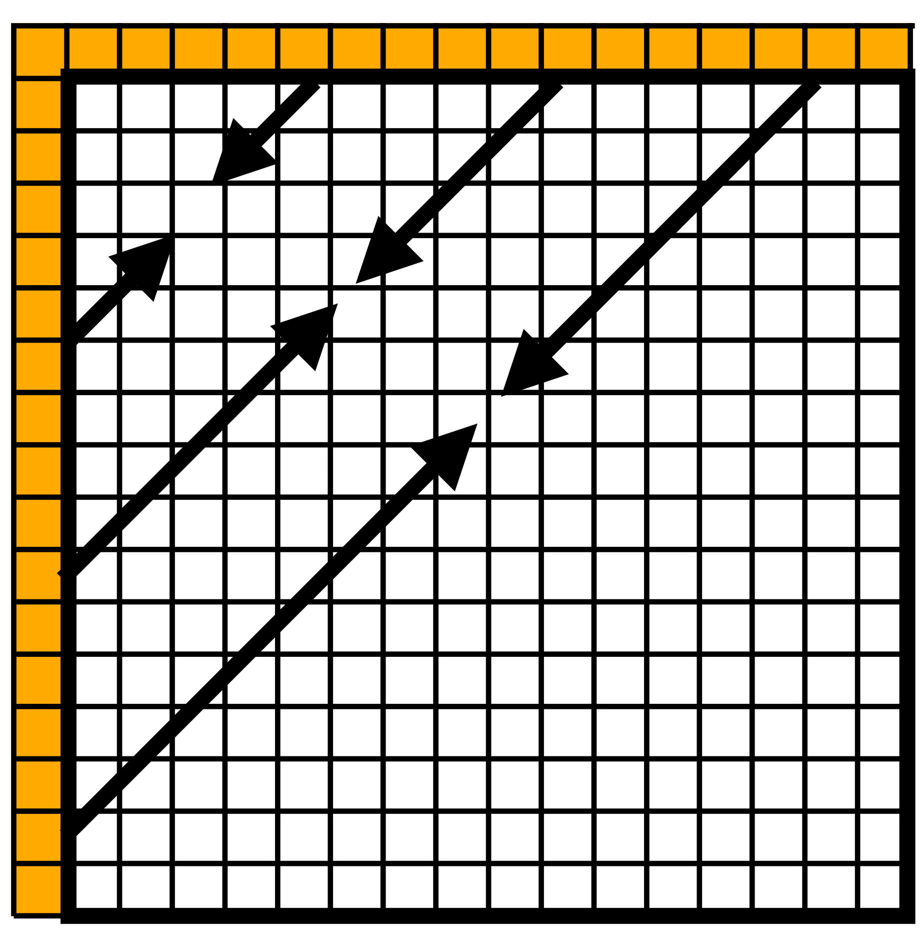
Mode 0: vertical



Mode 1: horizontal



Mode 2: DC



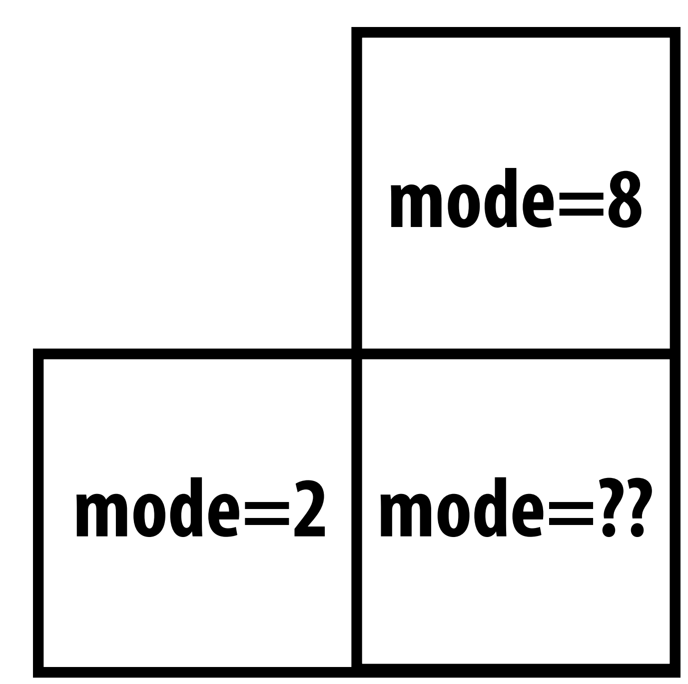
Mode 4: plane

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 is performed using the same four modes as intra_16x16 (except reordered as: DC, vertical, horizontal, plane)**

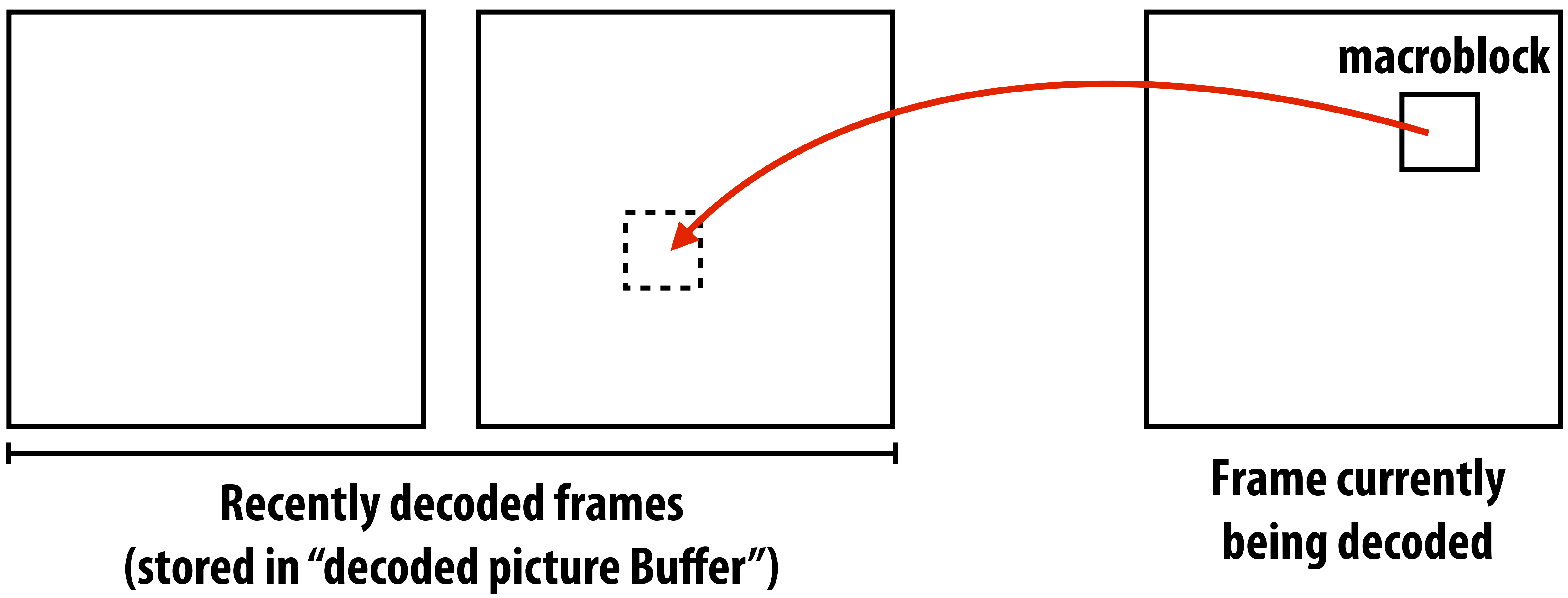
- **Intra-prediction scheme for each 4x4 block within macroblock encoded as follows:**
  - **One bit per 4x4 block:**
    - **if 1, use <u>most probable</u> mode**
      - **Most probable = lower of modes used for 4x4 block to left or above current**
    - **if 0, use additional 3-bit value `rem_intra4x4_pred_mode`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

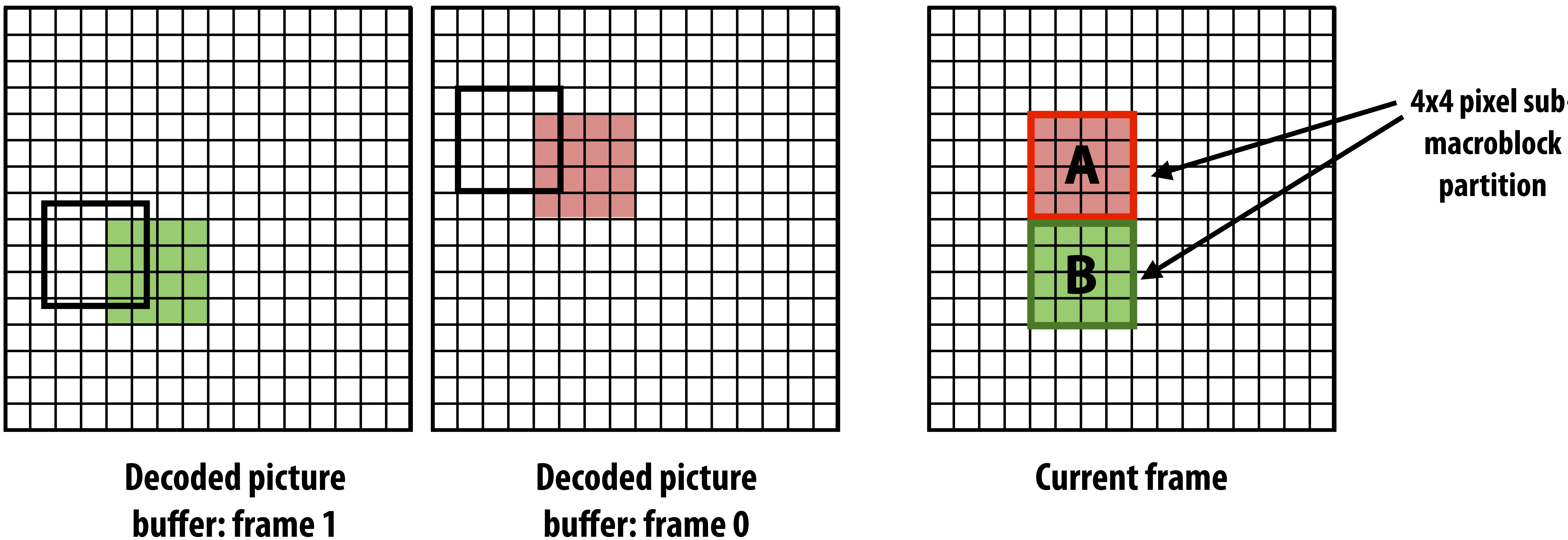
- Basic idea: current frame formed by translation of pixels from temporally nearby frames (e.g., object moved slightly on screen between frames)
  - "Motion compensation": use of spatial displacement to make prediction about pixel values



* Note: "previously decoded" does not imply source frame must come before frame in video sequence. (Can decode 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)

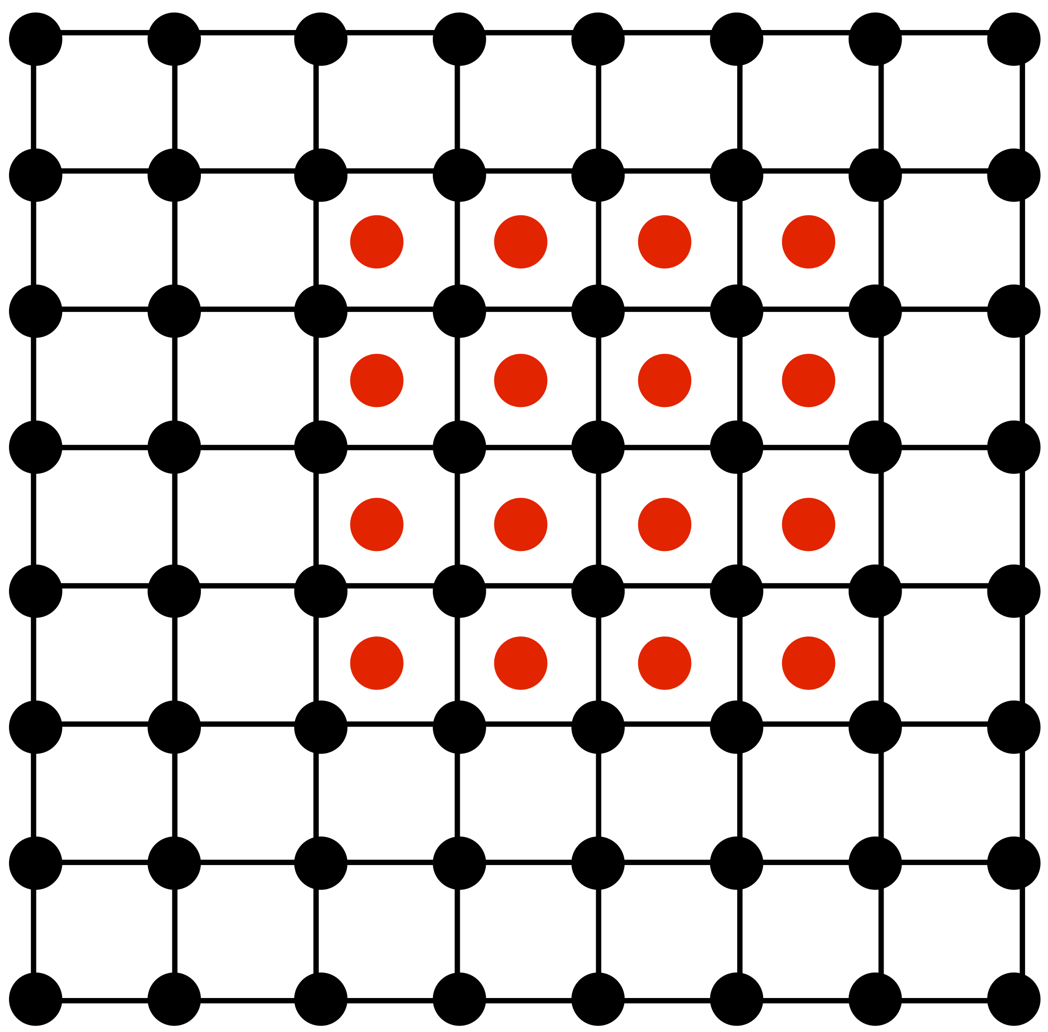


Block A: predicted from (frame 0, motion-vector = [-3, -1])

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

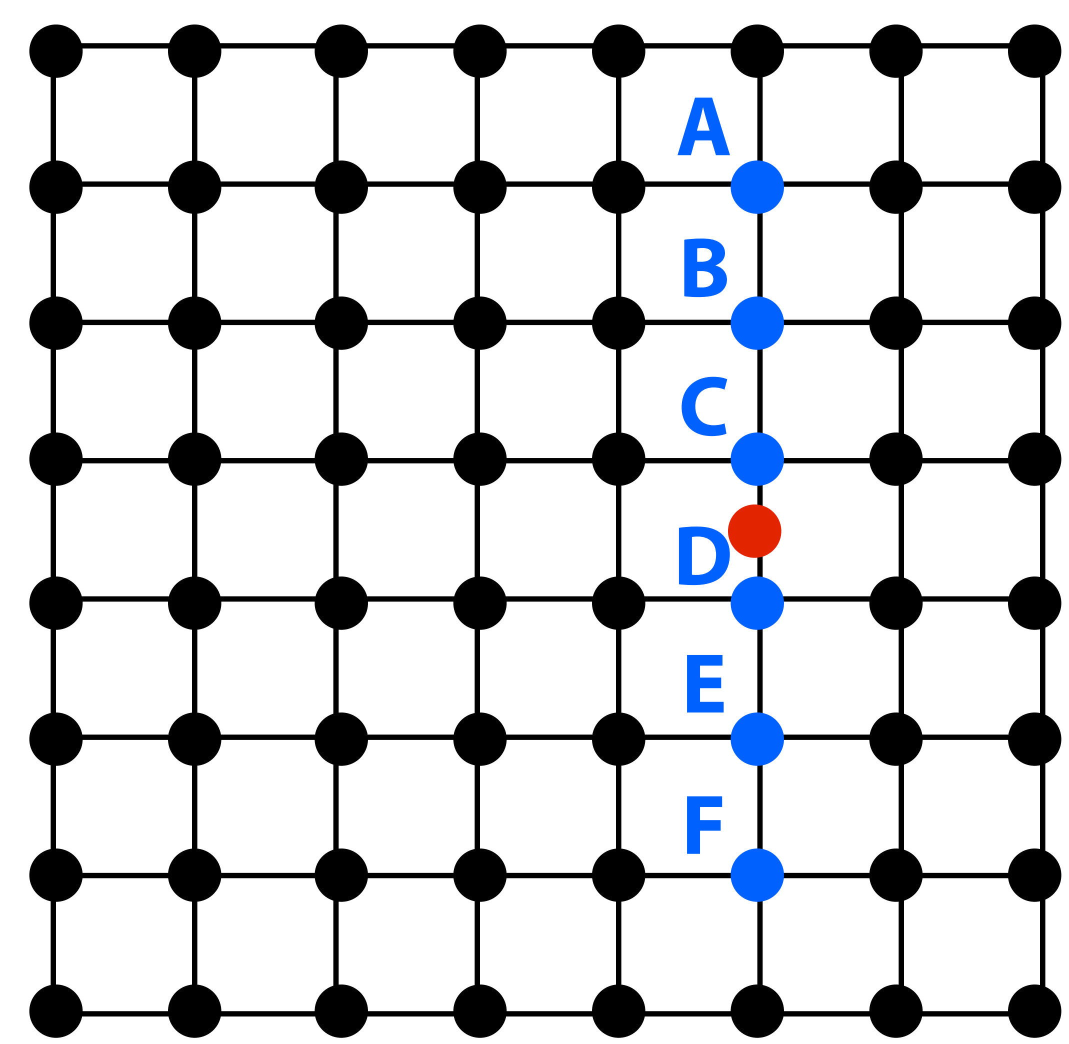
Notice non-integer motion vector

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 + 5C + 5D - 5E + F) / 32)`

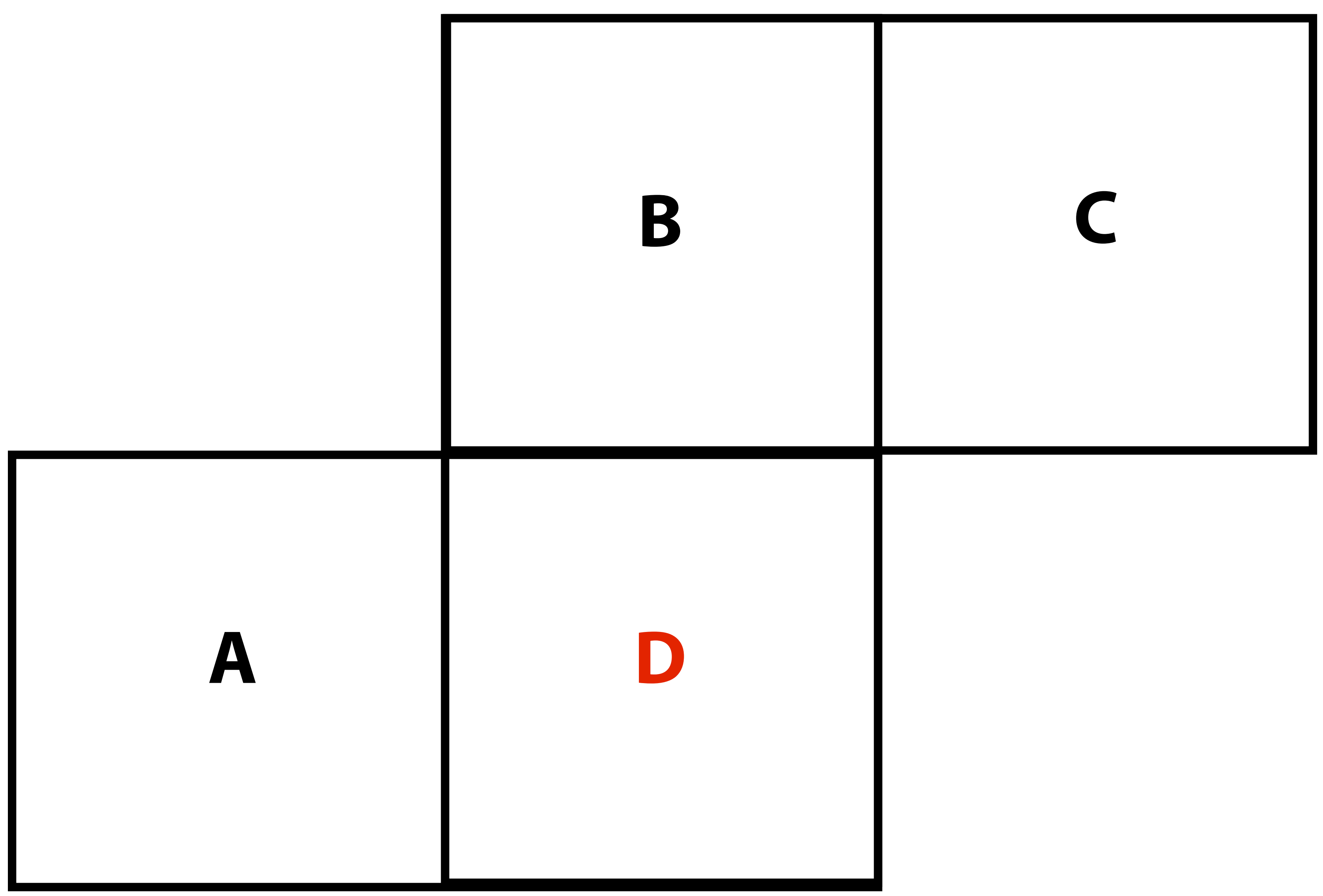
H.264 supports both 1/2 pixel and 1/4 pixel resolution motion vectors

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 requires significant amount of storage
- Solution: predict motion vectors from neighboring partitions and encode residual in compressed video stream
  - Simple example below: predict D's motion vector as average of motion vectors of A, B, C
  - More complex logic when partition sizes of neighboring blocks have different sizes



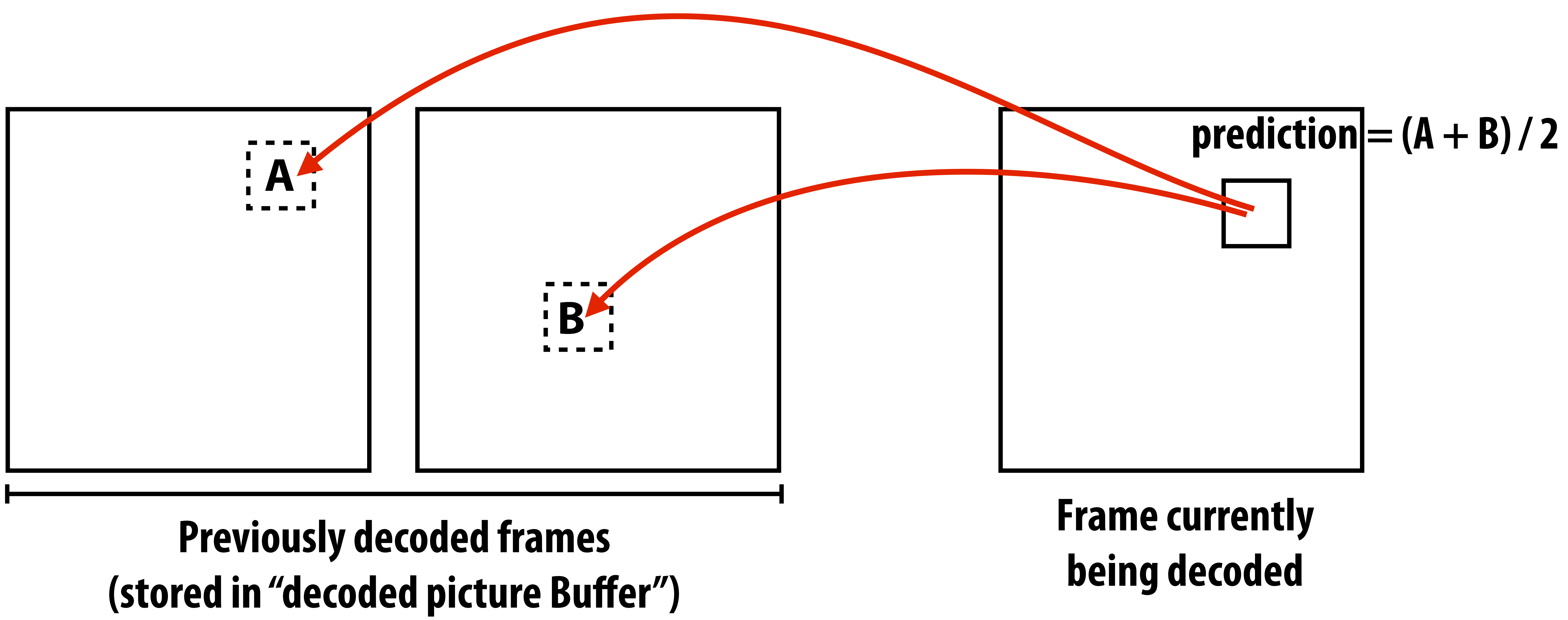
Question: what partition size is best?

- 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 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 superblock granularity encoding
  - After macroblock decoding is complete, optional perform smoothing filter across block edges.

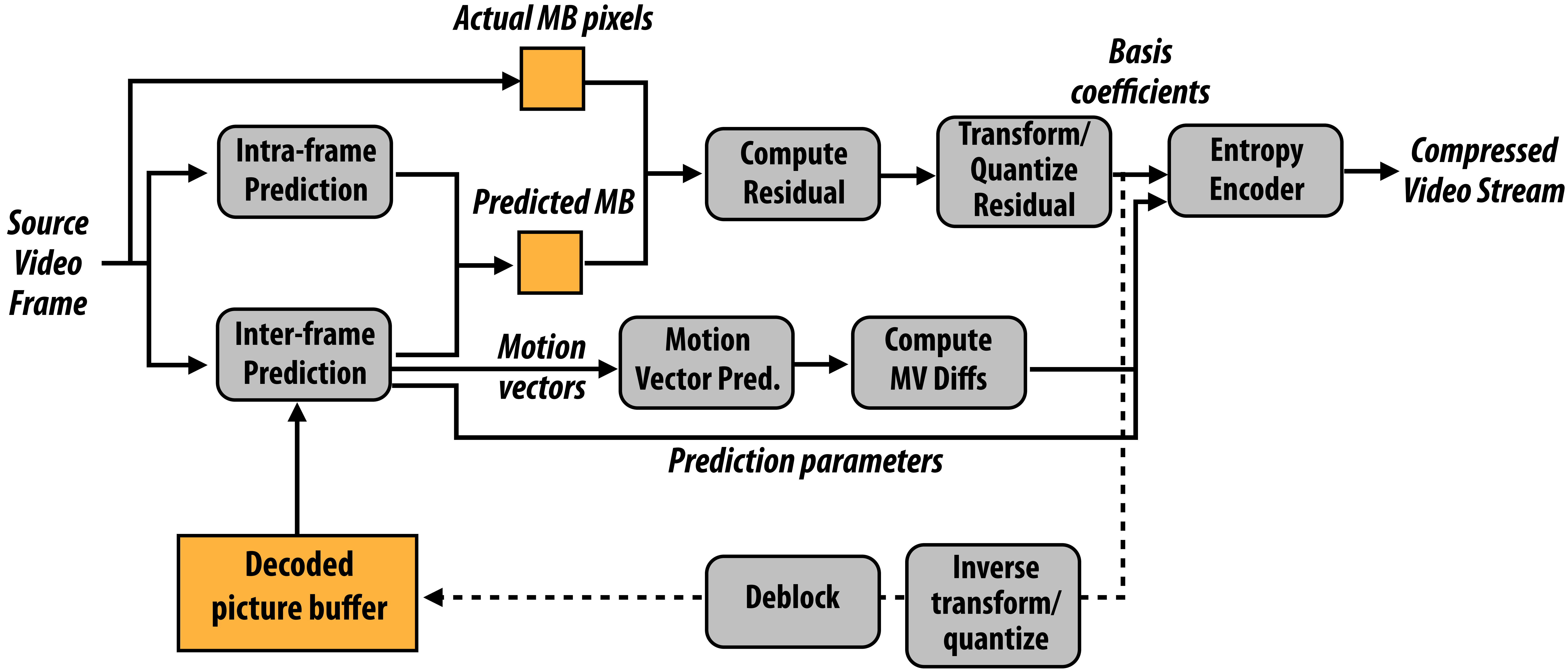
Putting it all together: encoding an inter-predicted macroblock

- Inputs:
  - Current state of decoded picture buffer
  - 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 macroblock partition (or sub-partition)
  - 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

Heavily coupled decisions

H.264/AVC video encoding

MB = macroblock
MV = motion vector

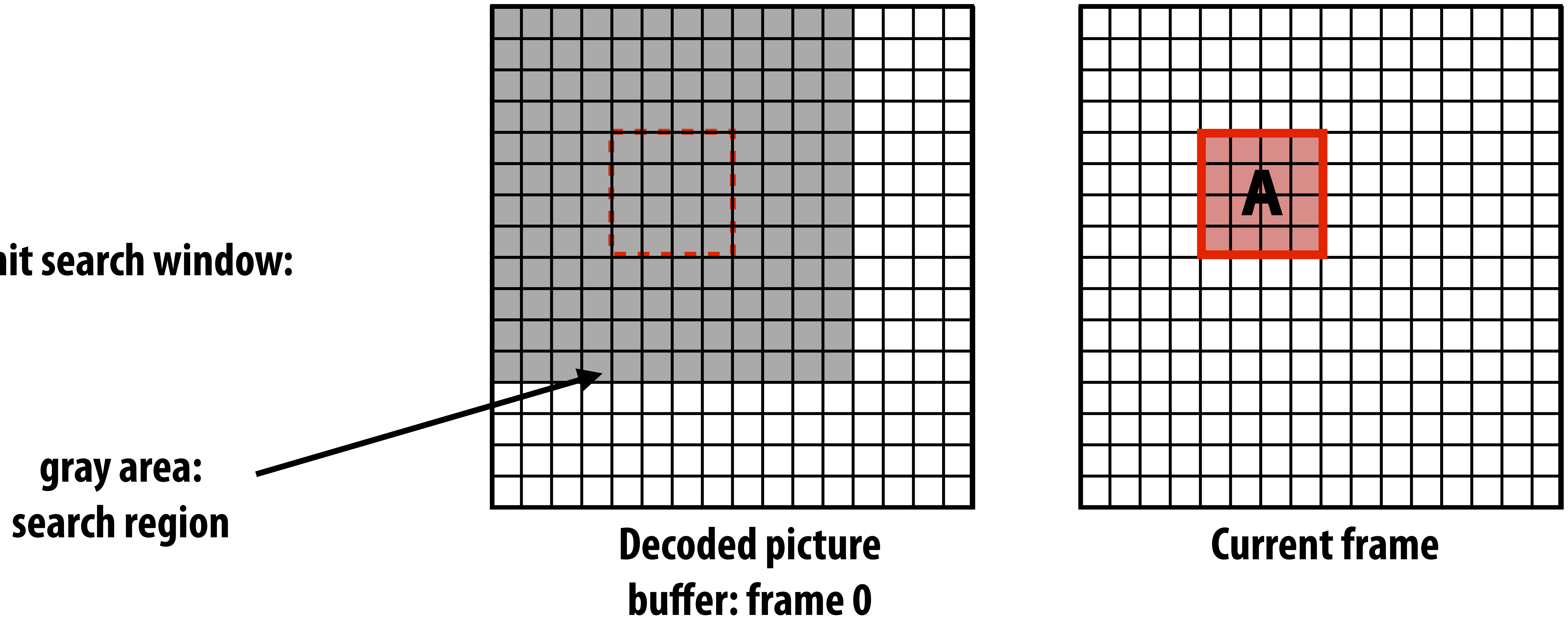


Credit: Figure derived from H.264 Advanced Video Compression Standard, I. Richardson, 2010

Motion estimation

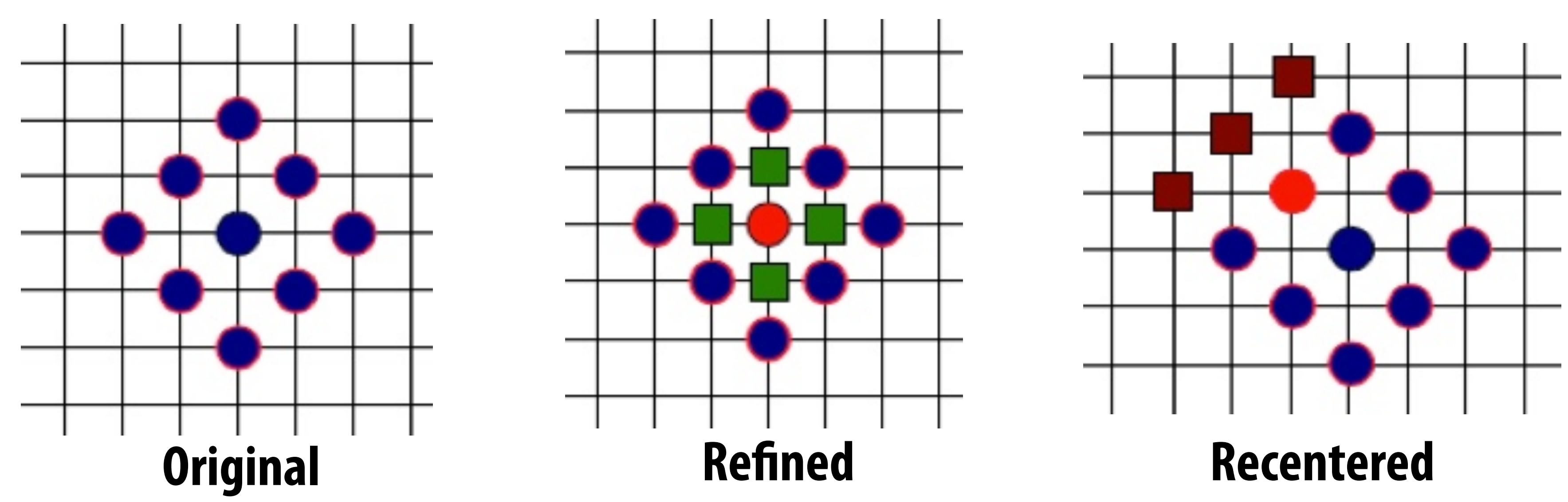
- Encoder must find reference block that predicts current frame's pixels well.
  - Can search over multiple pictures in decoded picture buffer + motion vectors can be non-integer (huge search space)
  - Must also choose block size (macroblock partition size)
  - And whether to predict using combination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

Limit search window:



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" compressed representation 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