Introduction to Video Encoding

INF5063

Håvard Espeland

September 4, 2012

History of MPEG

- Motion Picture Experts Group
  - MPEG1 work started in 1988, published by ISO in 1993
    - Part 1 Systems, Part 2 Video, Part 3 Audio, Part 4 Compliance Testing and Part 5 Software Simulation
    - MP2 and MP3 Audio
  - MPEG2 was published in 1996 together with ITU
    - 11 parts in total
    - Part 2 Video also known as H.262, used on DVD
    - Part 3 Audio (improved MP3) and Part 7 Audio (AAC)

History of MPEG

- MPEG4 was introduced in 1996
  - 27 parts in total, all known as MPEG4
  - Part 2 Visual (H.263): Simple Profile, Advanced Simple Profile (ASP)
    - Colloquially called MPEG4 (until recently)
    - Widely used in broadcasting, teleconferencing
    - Compresses much better than MPEG2
    - Often referred to by the name of the encoder (DivX, Xvid)
  - Part 3 Audio: AAC+, CELP, and more.

MPEG4

- Part 10 (H.264): Advanced Video Coding (AVC)
  - Introduced in 2003, and is still the most efficient (standardized) codec available
  - Up to about twice the compression of MPEG4 ASP
  - Used by Bluray, Rikstv, Youtube, and many others
  - Also referred to by the name of a specific encoder (x264)
  - 17 profiles, including Multiview High Profile (MVC) for stereo (3D) video, and Scalable High Profile (SVC) for adaptive streaming and trick-play functionality.

H.265 / HEVC

- High efficiency video coding: successor of H.264 / AVC.
- Draft standard, expected to be released in 2013.
- Provides ~ 20% bitrate reduction compared to AVC High Profile.
- Supports ultra high resolution (4320p or 8k).
- Will probably be preferred codec for UHDTV applications in the coming years.

MPEG Licensing

- Patent Pool created by MPEG-LA, an independent organization from MPEG
- Controls roughly 1500 patents related to H.264
- Incentive for industry players to contribute to standardization process
- Licensed under RAND terms (Reasonable and Non-Discriminatory terms)
  - Basically means that you pay per sold encoder and decoder

Web Video - *WebM*

- Google bought On2 in February 2010
  - Released the VP8 codec specification with a royalty free license in May 2010 together with an open-source implementation (libvpx)
  - Also released a container format (webm) based on Matroska (mkv) to distribute VP8 together with Vorbis.
- VP8 is very similar to H.264, but only supports a subset of the features
- Supported by all major browser vendors and mobile phones. Used by some streaming sites such as Youtube together with HTML5 <video>.

Video standards

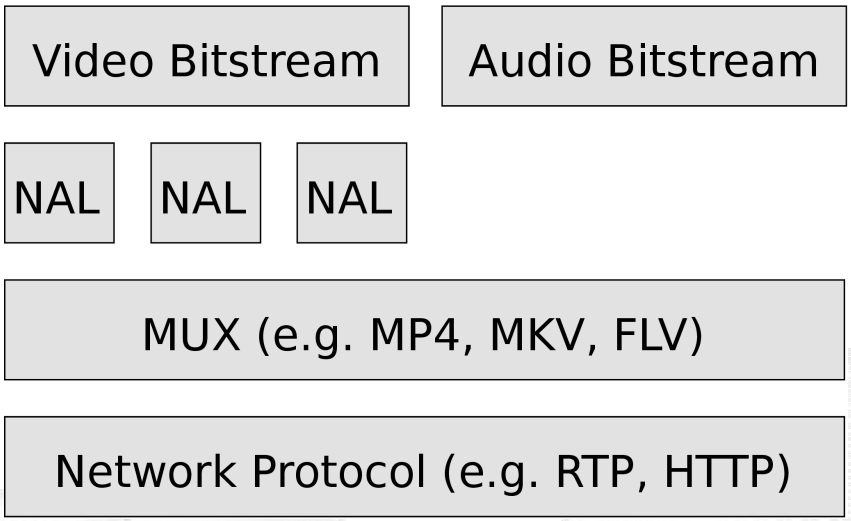
- The latest version of the H.264 specification has more than 600 pages, costs 294 CHF and can be bought from ISO/ITU.

- The VP8 codec is pending standardization under the IETF. Documents available for free from http://webmproject.org

- Video standards only describe *decoding* of the bitstream. The black art of *encoding* video is left as an exercise to implementers.

Video Format Stack

- Similarly to OSI, video delivery has multiple layers
- Codecs can be used in different MUXes and vice versa
- NALs are used in H.264 to help packetizing



JPEG Introduction

- "JPEG": Joint Photographic Expert Group (1992)

- International Standard:
  - For digital compression and coding of continuous-tone still images
  - Gray-scale and color

- Compression rate of 1:10 yields reasonable results
  - Lossless mode: reasonable compression rate approx. 1:1.6

- Independence of
  - Image resolution
  - Image and pixel aspect ratio
  - Color representation
  - Image complexity and statistical characteristics

Computer criminals

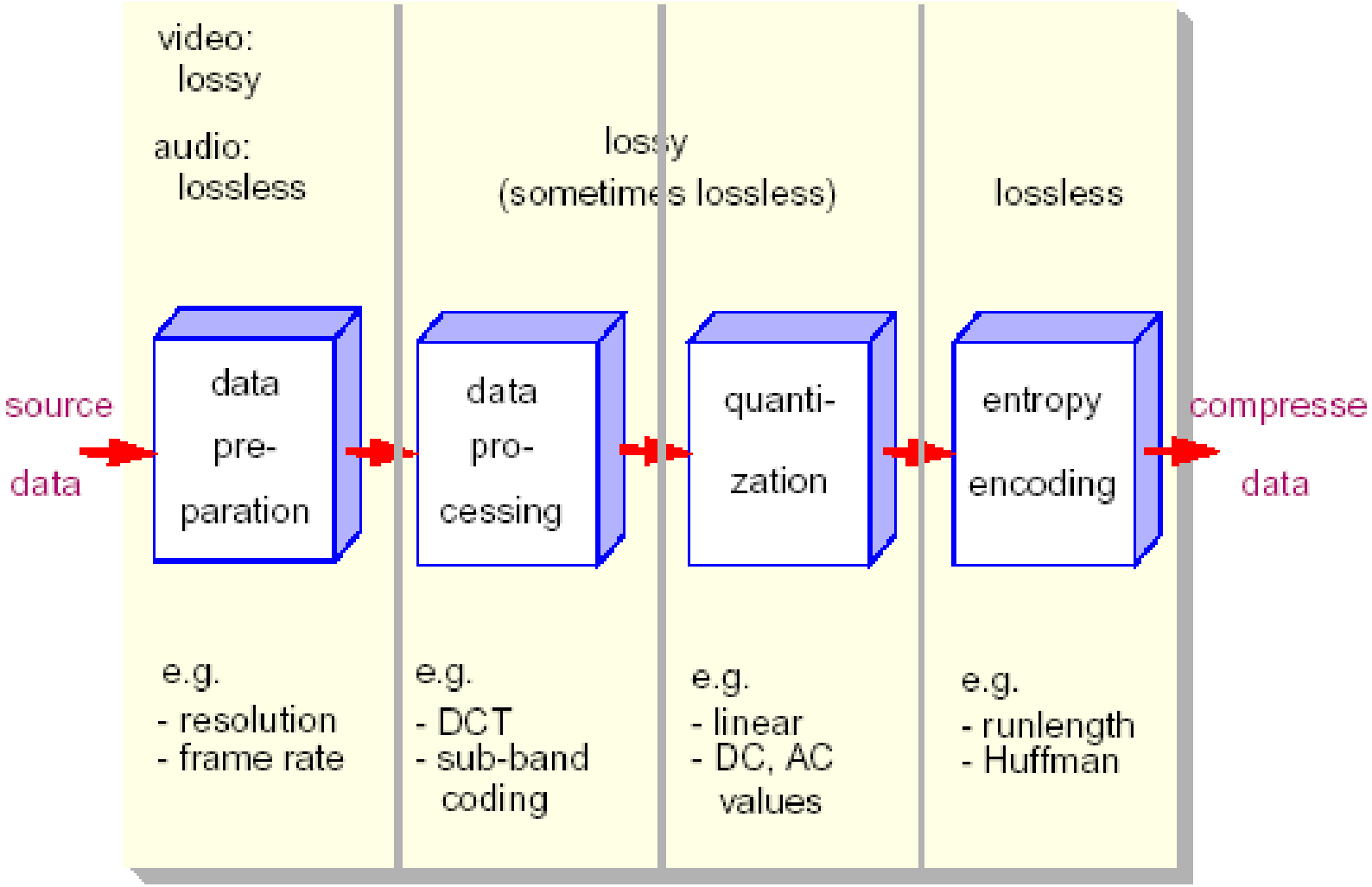
Computers will make the world of tomorrow a much safer place. They will do away with cash, so that you need no longer fear being attacked for your money. In addition, you need not worry that your home will be burgled or your car stolen. The computers in your home and car will guard them, allowing only yourself to enter or someone with your permission.

However, there is one kind of crime which may exist in the future – computer crime. Instead of mugging people in the streets or robbing houses, tomorrow's criminal may try to steal money from banks and other organizations by using a computer. The computer criminal works from home, using his own computer to gain access to the memories of the computers used by the banks and companies. The criminal tries to interfere with the computers in order to get them to transfer money to his computer without the bank or company knowing that it has been robbed.

Computer crime like this in fact exists already. However, it is very difficult to carry out a successful robbery by computer. Many computers have secret codes to prevent anyone but their owners from operating them. As computers are used more and more, it is likely that computer crime will become increasingly difficult to carry out.

Nevertheless, a computer criminal may succeed now and then and the detectives of the future will have to be highly skilled computer operators. There will probably be police computer-fraud squads, specially trained to deal with computer crime. Here you can see a squad arriving at the home of a computer criminal and arresting him as he makes a dash for it. He is clutching a computer cassette that contains details of his computer crimes, and the police will need this as evidence to prove that he is guilty.

Basic Encoding Steps



Color Conversion (RGB to YCbCr)

- The 24-bit **RGB** (red, green, blue) value is often converted to **YCbCr** consisting of
  - one luma component (Y) representing brightness:

$$Y' = Kr \cdot R' + (1 - Kr - Kb) \cdot G' + Kb \cdot B'$$

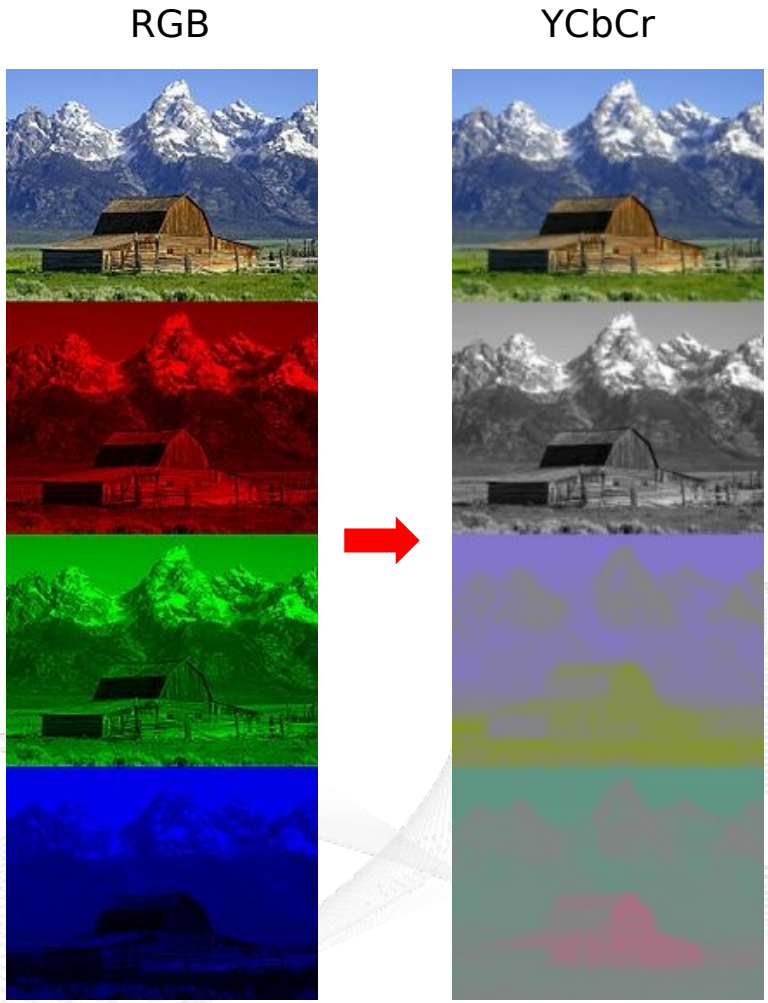
  - two chroma components representing color (Cb-BlueDifference and Cr-RedDifference), or of the Pb and Pr:

$$Pb = \frac{1}{2} \cdot \frac{B' - Y'}{1 - Kb} \qquad Pr = \frac{1}{2} \cdot \frac{R' - Y'}{1 - Kr}$$

  - Kb and Kr are ordinarily derived from the definition of the corresponding RGB space, e.g., Kb = 0.114 and Kr = 0.299 defined for standard-definition television
  - the prime (') symbols mean gamma correction is being used - resulting luma (Y) value will then have a nominal range from 0 to 1, and the chroma (Cb and Cr) values will have a nominal range from -0.5 to +0.5
  - in digital form, the results are scaled and rounded, and offsets are typically added

RGB to YCbCr

- Y image is essentially a greyscale copy of the main image;
- the white snow is represented as a middle value in both Cr and Cb;
- the brown barn is represented by weak Cb and strong Cr;
- the green grass is represented by weak Cb and weak Cr;
- the blue sky is represented by strong Cb and weak Cr.

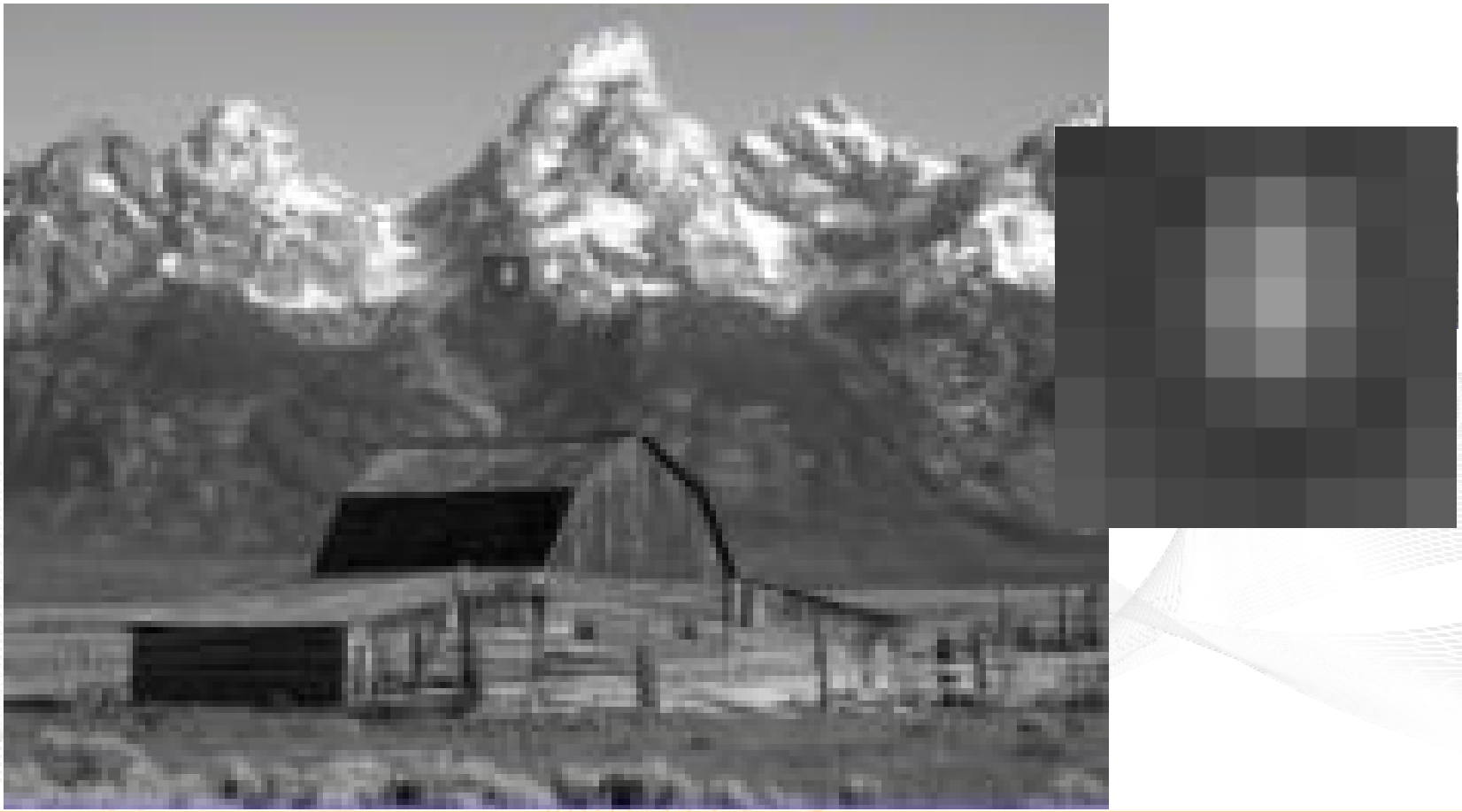


Split into 8x8 blocks

Each Y, Cb and Cr picture is divided into 8x8 blocks, number depends on resolution

Split into 8x8 blocks

Each Y, Cb and Cr picture is divided into 8x8 blocks, number depends on resolution



DCT Transformation

- Each 8×8 block (Y, Cb, Cr) is converted to a frequency-domain representation, using a normalized, two-dimensional DCT
  - two-dimensional DCT:

$$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

  - *Gu,v* is the DCT at coordinates *(u,v)*
  - *u* is the horizontal spatial frequency [0,8>
  - *v* is the vertical spatial frequency [0,8>
  - *gx,y* is the pixel value at coordinates (x,y)
  - α is a normalizing function:

$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

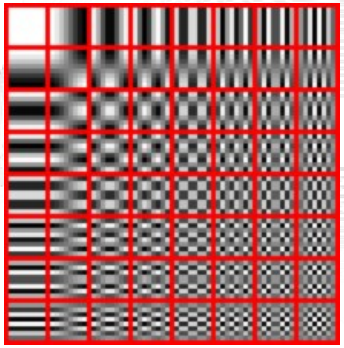
DCT Transformation

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} - 128 =$$

$$g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \begin{matrix} \xrightarrow{x} \\ \\ \\ \\ \\ \\ \\ \downarrow y. \end{matrix}$$

Apply DCT =

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \begin{matrix} \xrightarrow{u} \\ \\ \\ \\ \\ \\ \\ \downarrow v. \end{matrix}$$



Quantization

- The human eye
  - is good at seeing small differences in brightness over a large area
  - not so good at distinguishing the exact strength of a high frequency brightness variation
  - can reduce the amount of information in the high frequency components
  - simply dividing each component in the frequency domain by a known constant for that component, and then rounding to the nearest integer:

$$B_{j,k} = \text{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0, 1, 2, \cdots, N_1 - 1; k = 0, 1, 2, \cdots, N_2 - 1$$

where Qj,k is a quantization matrix

Quantization Example

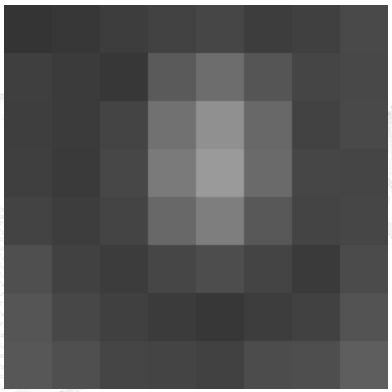
$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

($u \rightarrow$, $v \downarrow$)

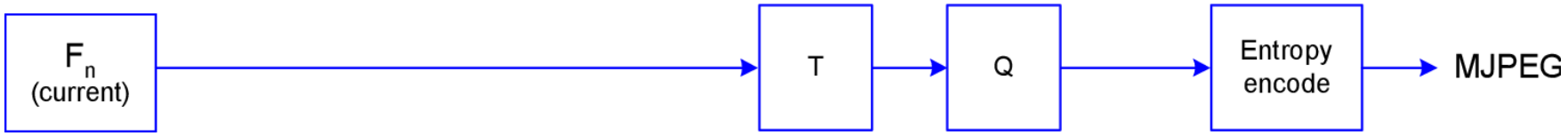
$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

$$B_{j,k} = \text{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0, 1, 2, \ldots, 7; k = 0, 1, 2, \ldots, 7$$

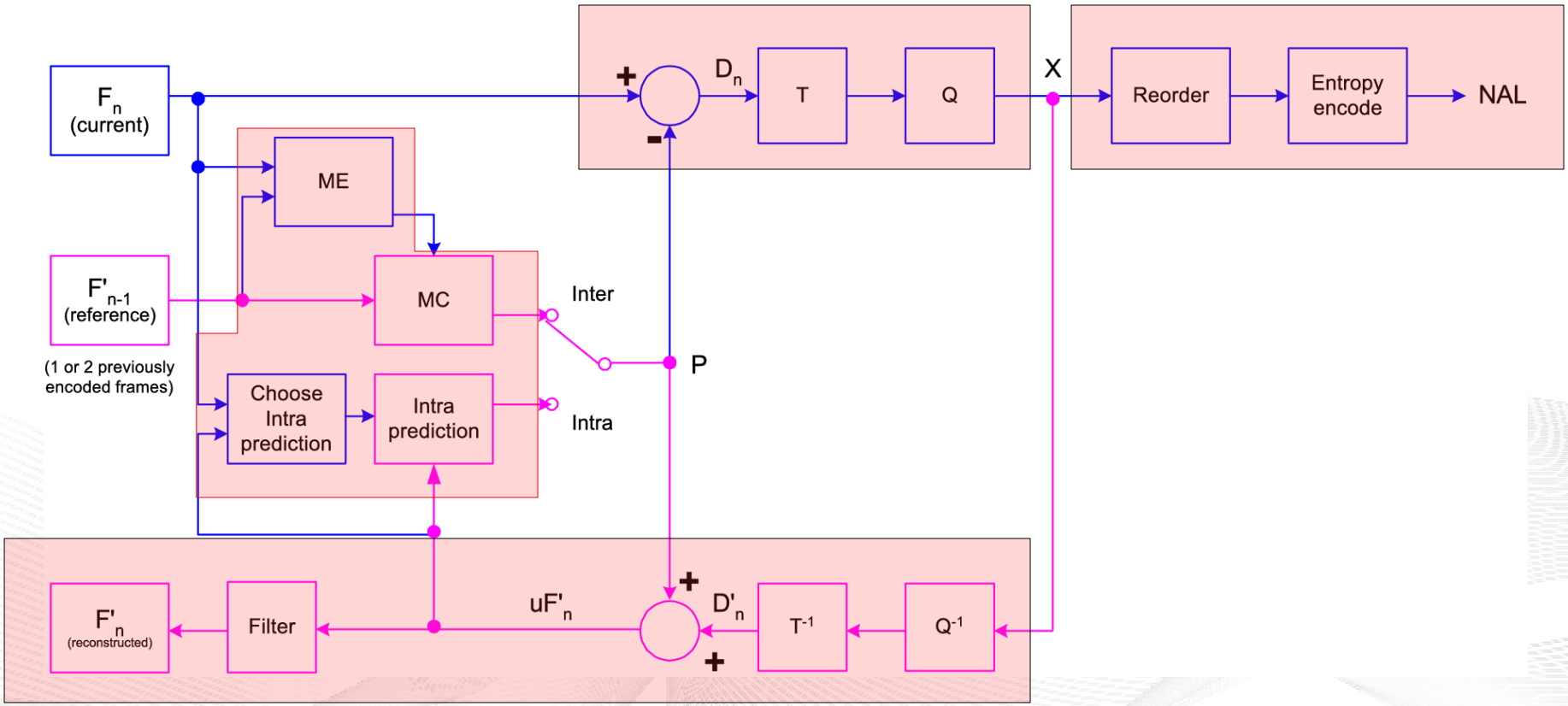
$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$



Motion JPEG Encoder Overview

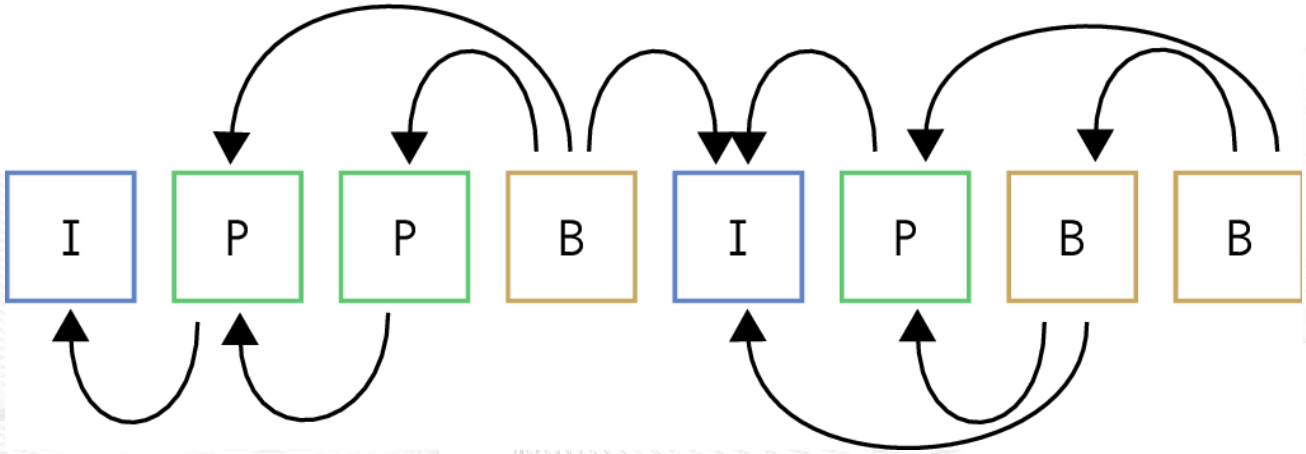


H.264 / VP8 Encoder Overview



Macroblock Types

- Macroblocks are 16x16 pixels, but can be subdivided down to 4x4 pixels. In H.264 they are of type I, P or B
  - Intra-MBs use Intra-prediction
  - Predicted MBs use Inter-prediction
  - Bi-Directional Predicted MBs use prediction both backward and forward in time

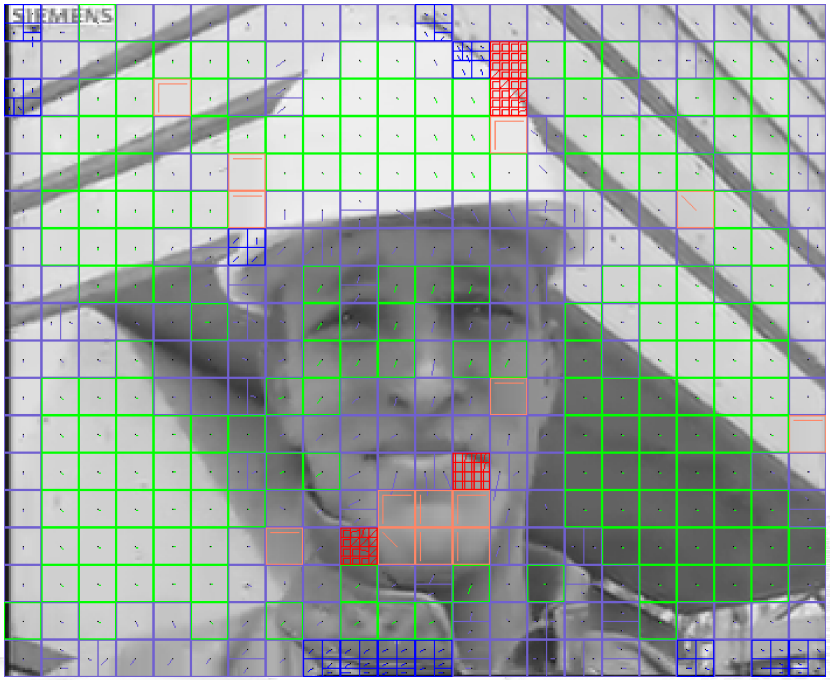


Frame types

- Traditional frame types are I-, P- and B-frames.
  - Intra-predicted frames can only use I-macroblocks.
  - Predicted frames can only use I- and P-macroblocks.
  - Bi-Directional predicted frames can use I-, P- and B-macroblocks
- VP8 does not have the concept of B-frames, but instead provides Alt-ref and Golden frames
  - Alt-ref frames are never showed to the user, and are only used for prediction

Inter-Prediction

- Predict a macroblock by reusing pixels from another frame
  - Objects tend to move around in a video, and *motion vectors* are used to compensate for this
  - H.264 allows up to 16 reference frames, while VP8 only supports 3 frames



Determining Prediction Modes

- Motion estimator tries as many modes and parameters as possible given a set of restrictions
  - The restrictions can be frame type, encoding time, heuristics, and possibly many more
- The different modes are evaluated with a cost function
- The estimator often use a two-step process, with initial coarse evaluation and refinements
- Refinements include trying every block in the area, and also using sub-pixel precision (interpolation)
- Many algorithms exist designed for different restrictions

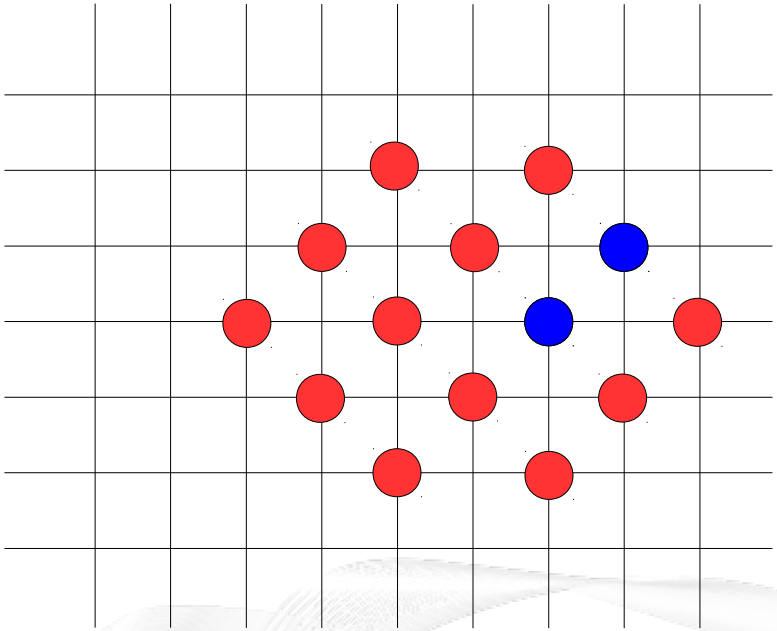
Cost Functions

- Typically Sum of Absolute Differences (SAD) or Sum of Absolute Transformed Differences (SATD)

- SATD transforms the sum with a Hadamard transformation
  - More accurate than SAD

$$SAD = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |I_{i,j} - T_{i,j}|$$

Diamond Motion Estimation Pattern



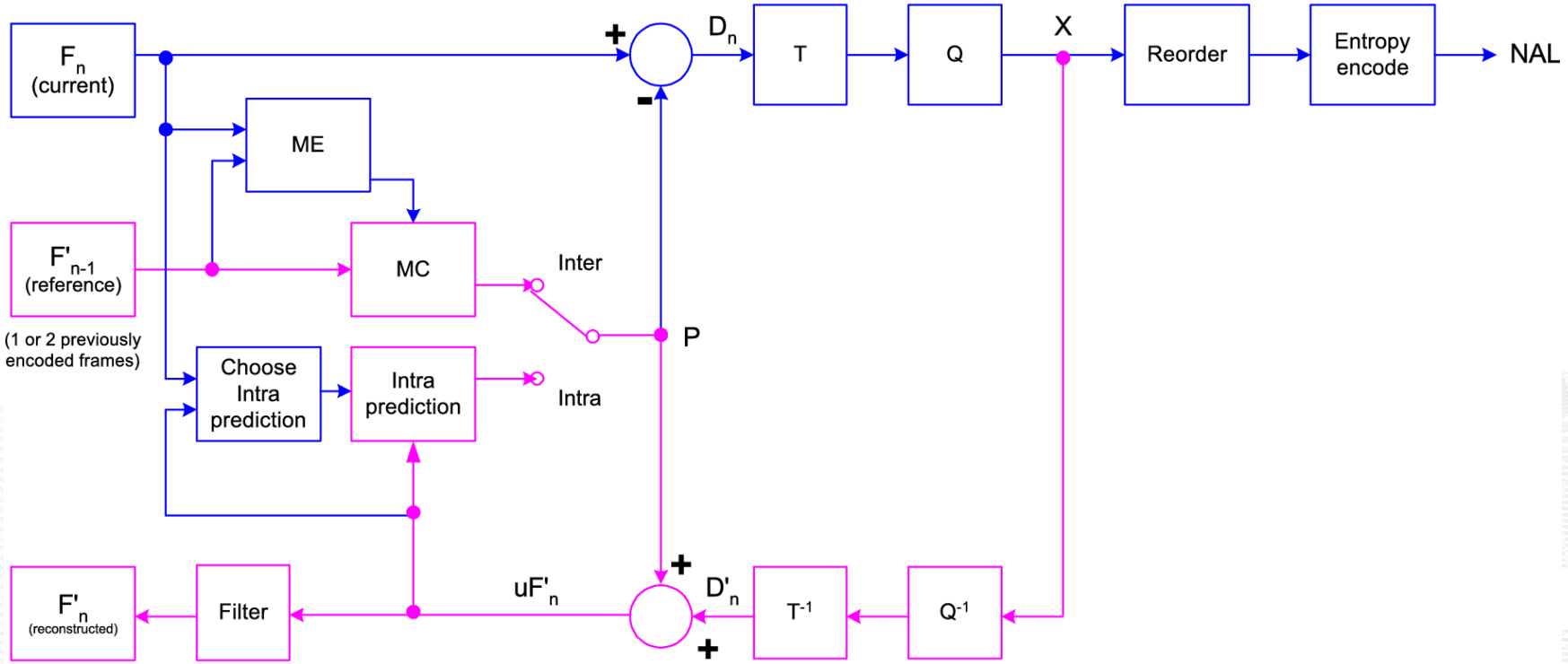
Motion Compensation

- When the best motion vector has been found and refined, a predicted image is generated using the motion vectors

- The reference frame can not be used directly as input to the motion compensator
  - The decoder never sees the original image. Instead, it sees a *reconstructed* image, i.e. an image that has been quantized (with loss)

- A reconstructed reference image must be used as input to motion compensation

H.264 / VP8 Encoder Overview



Intra-Prediction

- Predict the pixels of a macroblock using information available within a single frame
- Typically predicts from left, top and top-left macroblock by inter- or extrapolating the border pixel's values
- Many prediction modes available, e.g. horizontal, vertical, average

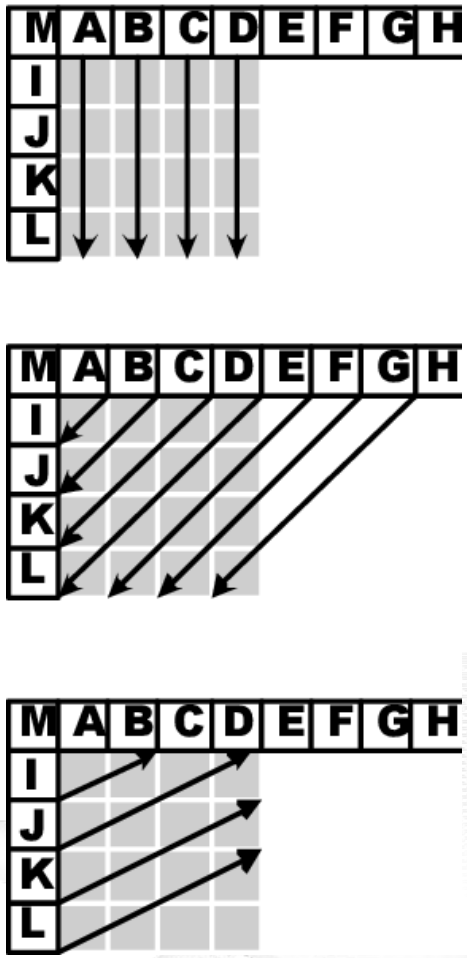
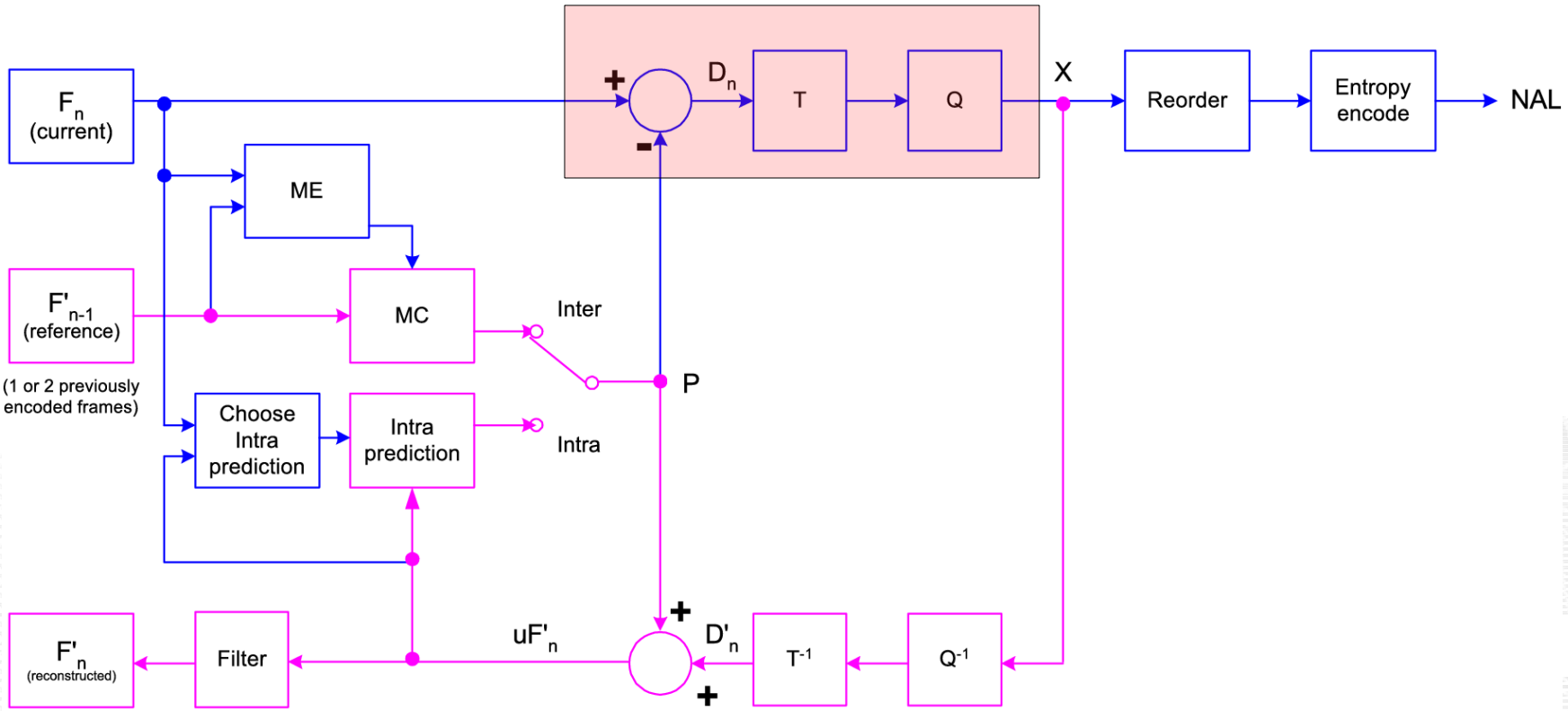


Figure from vcodex.com

H.264 / VP8 Encoder Overview



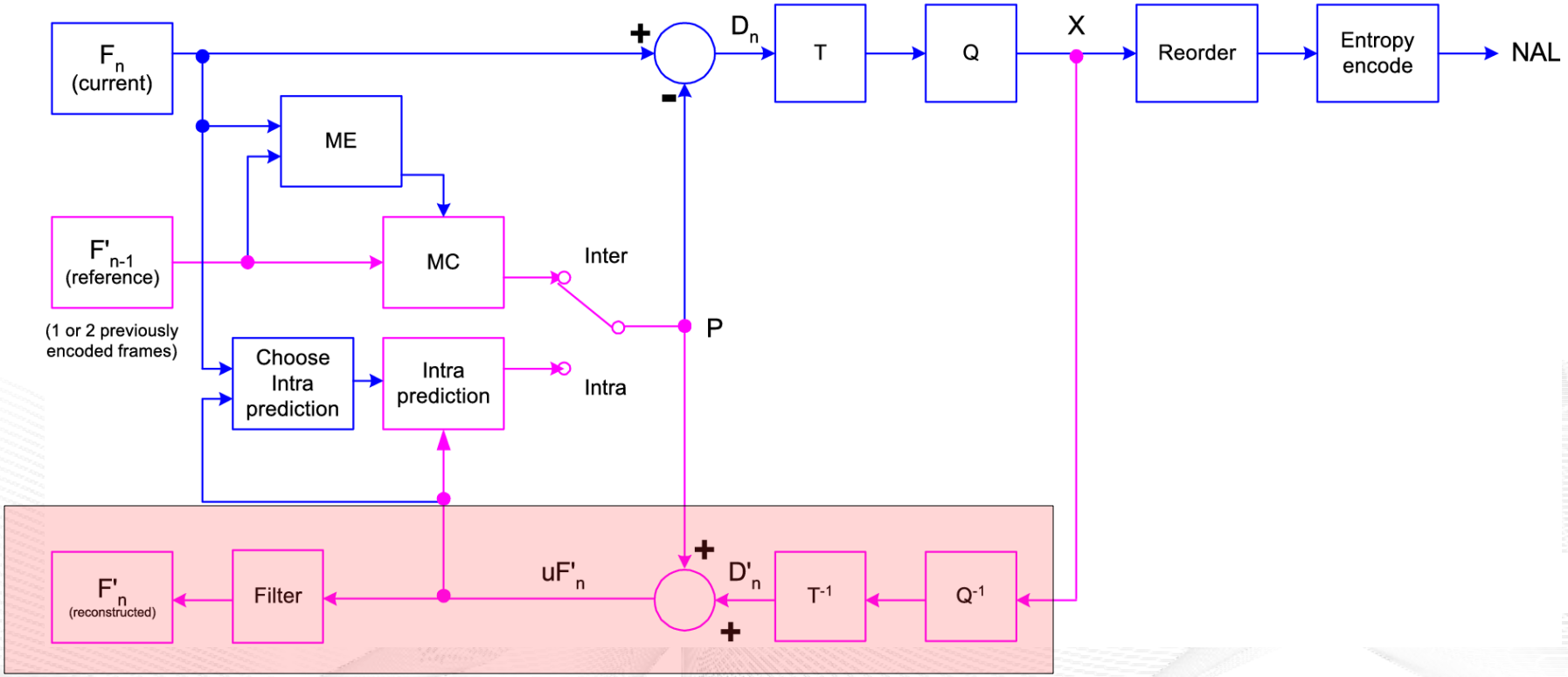
Residual Transformation

- The pixel difference between the original frame and the prediction is called residuals
- Since the residuals only express the difference from the prediction, they are much more compact than full pixel values such as in JPEG.
- Residuals are transformed using DCT (H.264) or a combination of DCT and Hadamard (VP8)
- Other alternatives exist such as Wavelets (Dirac)
- VP8 and H.264 use 4x4 DCT functions to reduce computation complexity

Quantization and Rate Control

- Divide the transformed residuals (coefficients) with a quantization matrix
  - Decimates the precision of the pixel frequencies, i.e. the lossy part of a video codec
  - The matrix is often scaled with a quantization parameter (QP) to a desired level of reduction
- Each frame is given a bit budget and QP is adjusted to match the budget

H.264 / VP8 Encoder Overview

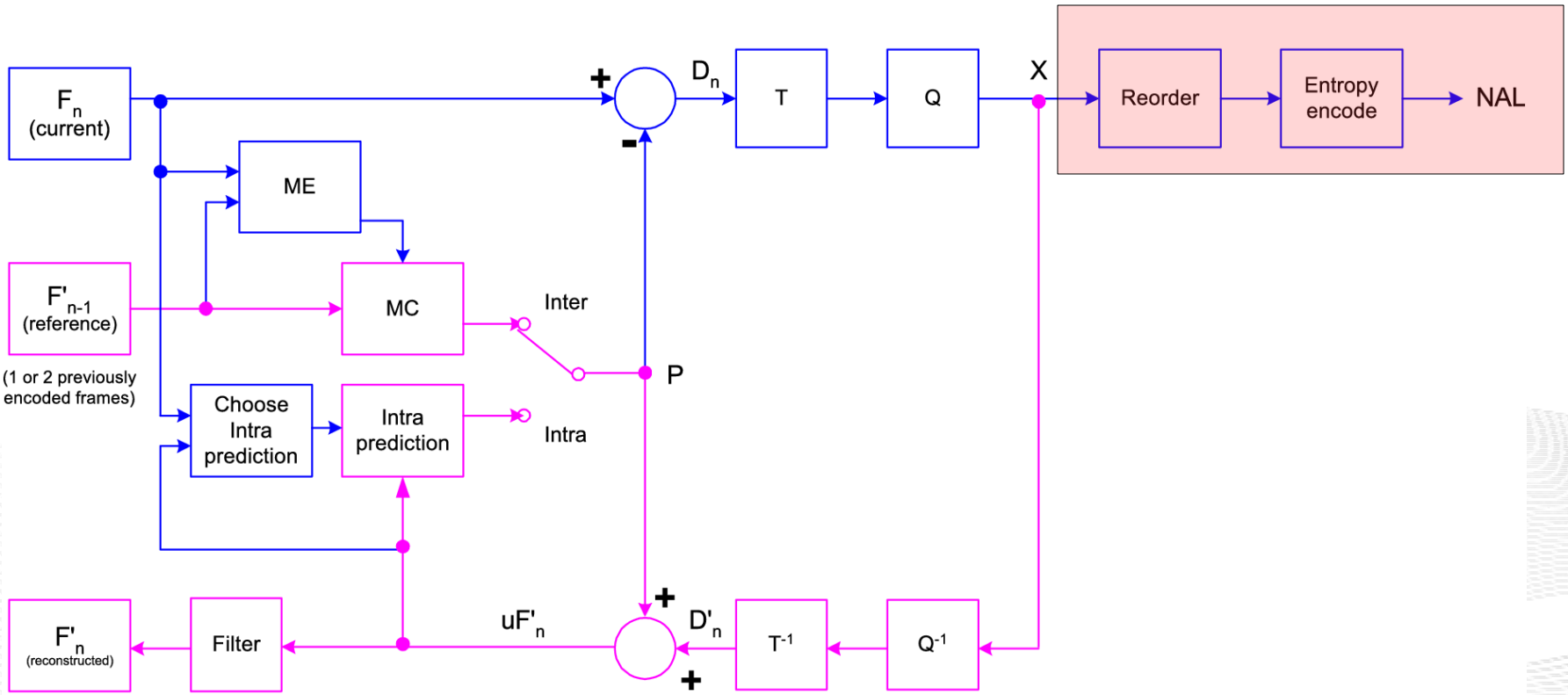


Frame Reconstruction

- The motion compensator requires as input the same reference frame as the *decoder* will see
- De-quantize and inverse transform the residuals and add them to our predicted frame
  - The result is the same *reconstructed* frame as the decoder will receive

- To increase visual quality, H.264 and VP8 use a deblocking filter to remove hard edges from the macroblocks

H.264 / VP8 Encoder Overview



Frame Reordering

- In H.264, B-frames may reference *future* frames
- The frames are reordered in such a way that frames used for predicting other frames are decoded first
- In practice, this means that decoding order and display order of frames may differ
- VP8 does not have this concept, but can instead use alt-ref frames for prediction

Entropy Coding

- The motion vectors, intra-predictors, encoder parameters and residuals must be stored somehow
- The entropy coding process is lossless, and removes redundancy from the output bitstream
- Many alternatives exist
  - Variable Length Coding with RLE (MPEG4 ASP)
  - Arithmetic Coding (H.264 and VP8)
  - Exp-Golomb Coding (VC-1)
- The symbol probabilities change over time and are continously updated by the encoder
- The symbol probabilities may depend on the context

Variable Length Coding - Huffman

- Since the literals we want to store varies in length, VLC prepends the literal with a symbol that represents the *length* of the literal instead of storing the maximum length bits

To store an uint8_t v1 = 7;

bit_width(7) = 3

Output: 100 111 (6 bits)

Store uint8_t v2 = 0;

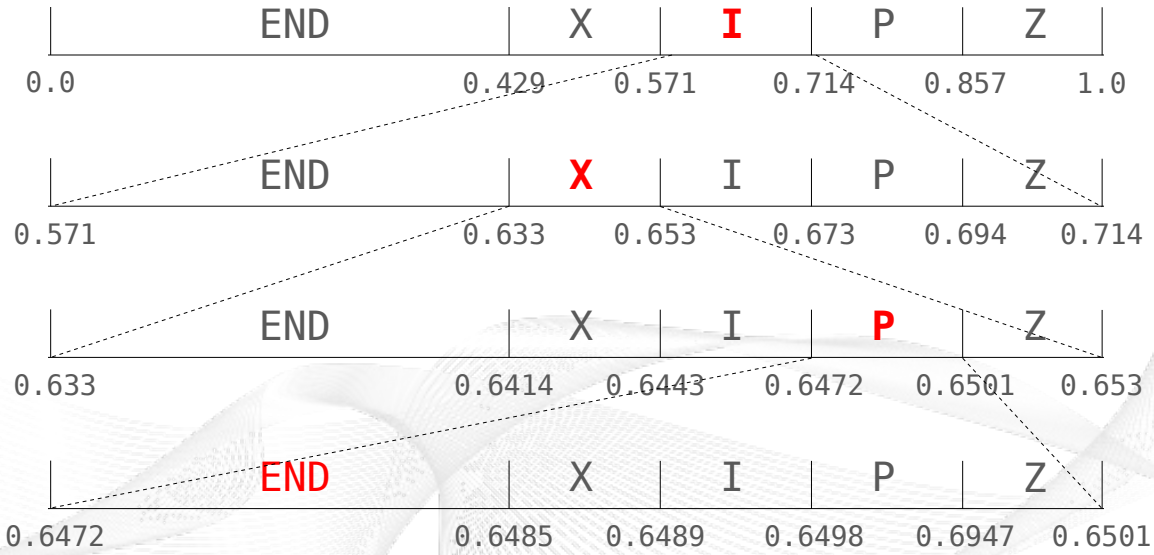
bit_width(0) = 0

Output: 00

Prefix Code	Bit Length
00	0
010	1
011	2
100	3
101	4
110	5
1110	6
11110	7

Arithmetic Coding

- Encodes the entire message into a single number between 0.0 and 1.0.
- Allows a symbol to be stored in less than 1 bit (!)



Encode IXP as a number between 0.6472 and 0.6485, e.g 0.648 = 1010001000

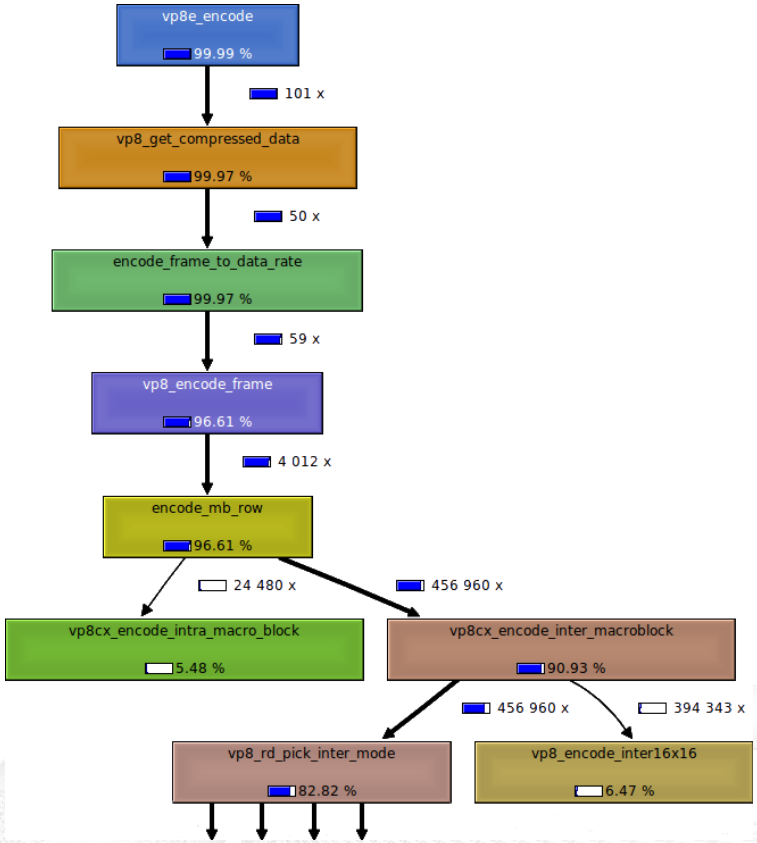
Error Handling

- Forward Error Correction
  - Add redundancy to regenerate original data content in case of lost or corrupt data.
    - Typically Reed Solomon or simpler Hamming Codes
- Periodic Intra Refresh
  - Refresh only a subset of macroblocks with *Intra Blocks*, and do so every frame
  - Alternative to a full I-frame
  - More graceful error handling
    - At the event of lost or corrupt data, data is refreshed gradually, instead of waiting for a full I-frame

Parallel Encoding

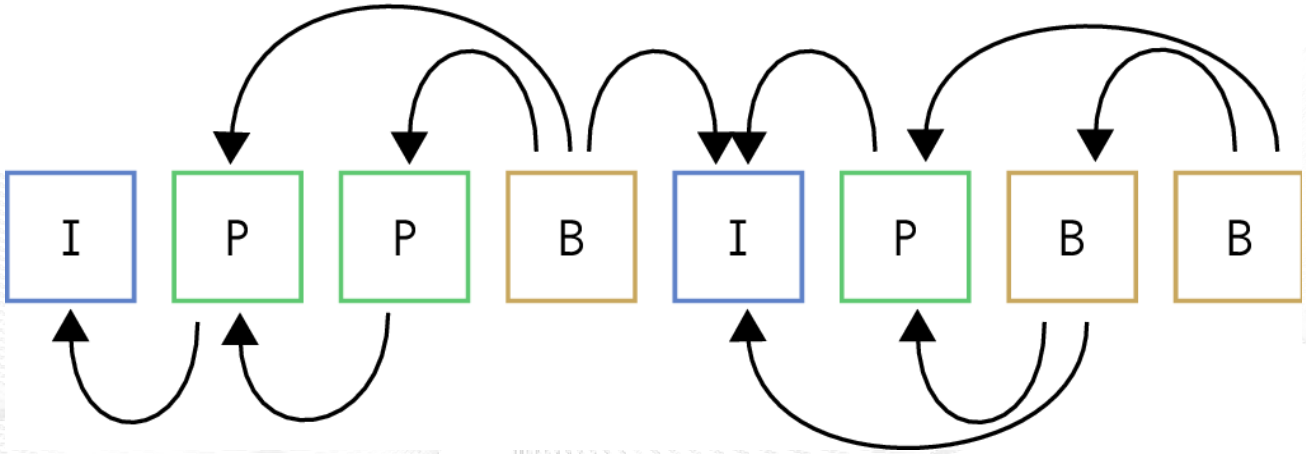
- Many approaches available both for Intra and Inter prediction
- Some of these give up compression efficiency for increased parallelism
- Pipeline approaches do not combine well with real-time encoders

What parts should be optimized?



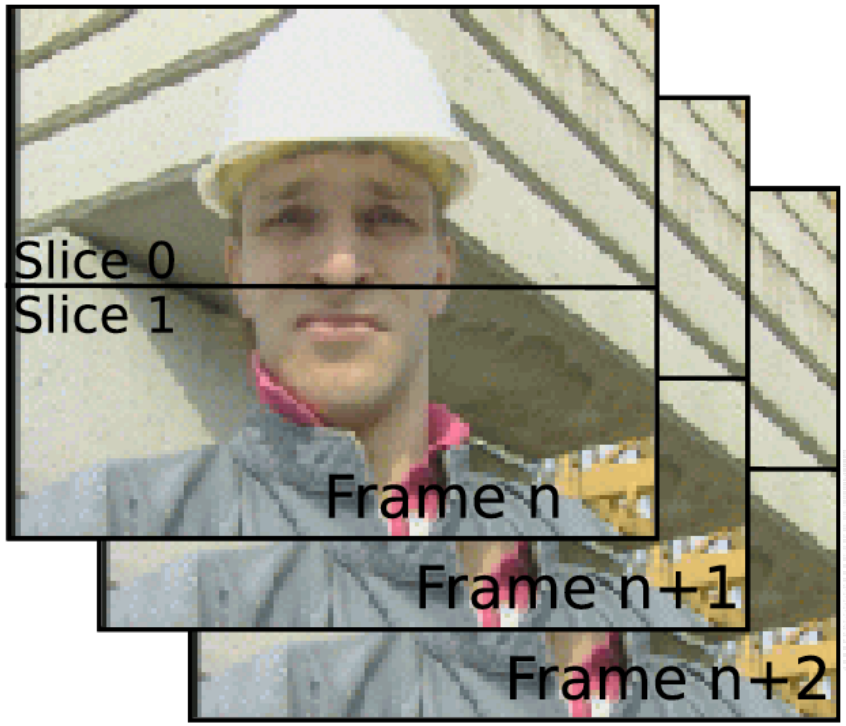
GOP-based Approach

- Delay pictures until there are multiple keyframes within the encoder pipeline
- Intra-frames do not depend on previous frames, and thus can be used as starting points for multiple encodes
- Adds a considerable pipeline to the encoder



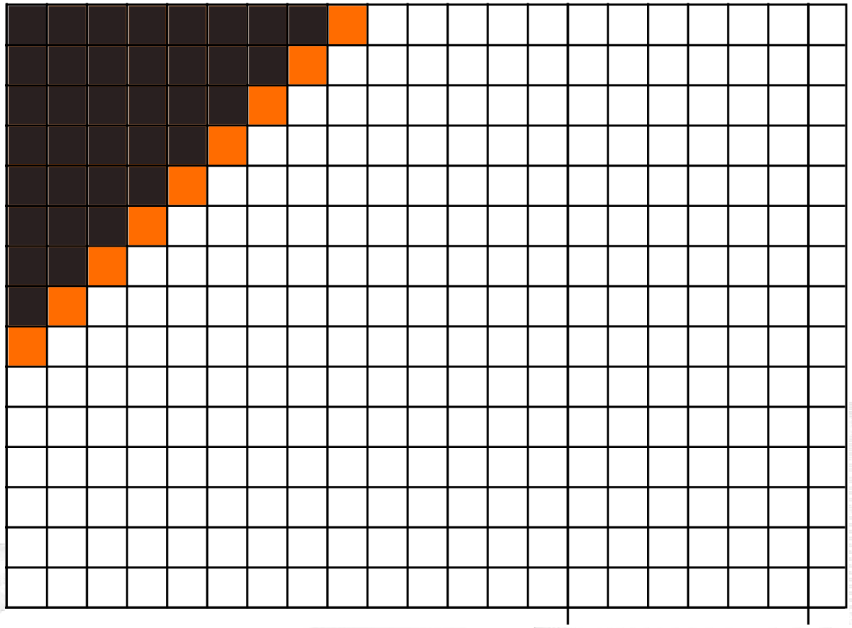
Slice-Based Approach

- Split every frame in one or more slices
- Supported by H.264
- Slices are completely independent – one may not predict across borders
- Severely hurts compression efficiency
- Typically used by the Apple H.264 compressor



Triangle Intra-Prediciton

- MBs depend on the left, top and top-left MB relative to its own position
- As soon as those dependencies are satisfied, the encoder can process in parallel without sacrificing encoder efficiency



Original-Frame Approach

- Both Intra- and Inter-prediction can use the original frame instead of the *reconstructed* frame when evaluating prediction modes

- However, when doing the actual motion compensation and intra-prediction, the encoder *must* use the reconstructed frame – if not there will be a drift between the encoder and decoder

- This approach reduces the quality of predictions

MV-Search Within a Frame

- When the reference frame has been fully reconstructed, the Motion Estimator tries for every macroblock to compensate for motion
- Since every motion vector is independent in the estimator, all MVs in a single frame can be search for in parallel
- When using optimized MV search patterns that takes advantage of the nearby block's MVs, this must be done in a manner similar to triangle intra-prediction
- Does not reduce encoding efficiency

MV-Search Across Frame Boundaries

- Reconstruct part of the frame as soon as the ME is finished with a macroblock
- When a large enough area has been reconstructed of the reference frame, the next frame's ME can start searching MVs that can only be found in this area
- The same technique can be used on multiple frames simultanously
- Requires synchronization on macroblock level
- Advanced technique with good performance
  - Does not reduce encoding efficiency
  - Adds a pipeline to the encoder – but it does not have to be deep. A few frame's delay can be acceptable for realtime encoding

Video Quality Assessment

- Evaluating video quality is a research field by itself
- Usually requires a panel of subjects that rate which version is *best*
  - Subjects have different preferences for artifacts and quality reductions
- Objective measurements give a rough number which says something about the difference between the original frames and the reconstructed frames
  - Typically SSIM or PSNR
  - A shell script for finding PSNR values of YUV frames can be found in the mplayer source tree under TOOLS/

Conclusion

- Video encoding is mainly about trying (and failing) different prediction modes limited by user-defined restrictions (resource usage)

- The “actual” encoding of the video when the parameters are known usually accounts for a small percentage of the running time

- Any (reasonable) codec can produce the desired video quality – what differs between them is the size of the output bitstream they produce