

VO Videoverarbeitung

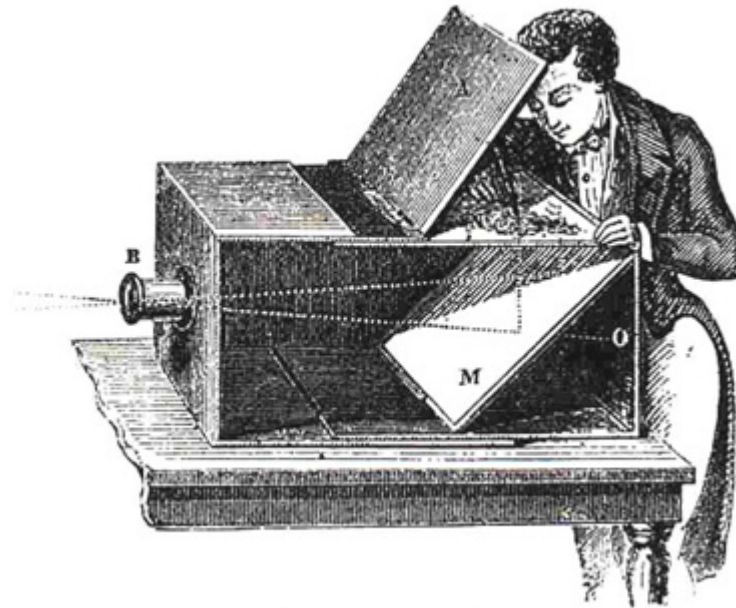
The Insides of Video Coding

Gastvortrag: Florian Seitner
seitner@ims.tuwien.ac.at

Overview

- Evolution of video coding
- Video coding techniques
- Real-time issues
- Parallelization

When did it start...?



When did it start...?

“Interaction” between the short-comings of the human eye and technical advances

- Ancestor of the modern slide projector
- China, 2nd century A.D.
- Projected paintings



“Laterna Magica”
(Magic lantern)

When did it start...?

“Interaction” between the short-comings of the human eye and technical advances

- Rotating cylinder with slits
- Images “blur” together
- Objects appear thinner than they are!

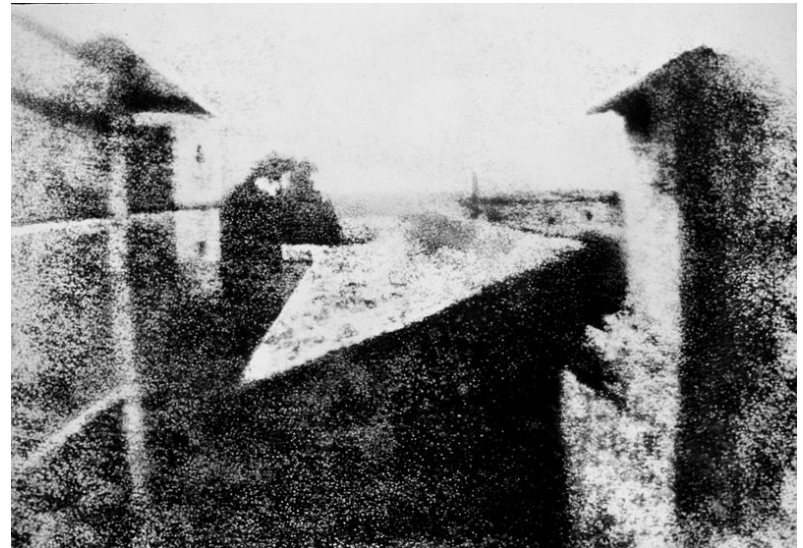


Zoetrope¹, 1834

Chemical film (1860s)

- Celluloid film
- Still pictures
- Black and white

8 hours exposure time!
Sun-light?



(Nicéphore Niépce, 1826)

Motion Capture Camera (1880s)

- “Silent era”
- Pianist, orchestra



Roundhay Garden Scene
(1888, Louis Le Prince)

“Talking pictures” (1920s)

- Early days of hollywood
- Attach synchronized(!) soundtrack to film
- F.W. Murnau
(*Nosferatu, The last laugh*)
- Charles Chaplin, Buster Keaton
- ...
- Color television (1960s)



Nosferatu, 1922
(Max Schreck as Count Orlock)

Video Evolution

- Projection of paintings
- Impression of movement (paintings)
- Still images of the “real” world
- Moving images
- Sound
- Color

The Digital Age (1980s)

- Projection of paintings
- Impression of movement (paintings)
- Still images of the “real” world
- Moving images
- Sound
- Color
- Video storage & transmission
- Information reduction
- Complexity vs. quality
- New applications

Information Reduction

- Removal of redundant information
 - No information loss
 - Information theory
 - Spatial & temporal similarities
- Removal of irrelevant information
 - Not noticeable by human observer
 - Exploit shortcomings of the Human Visual System (HVS)

The Human Visual System (HVS) (1)

- Reconstruct the spatial composition of a scene from overlapping, light, shade, motion and experience.
2-D images → 3-D scene
- Limited temporal resolution
 - Sampling with 25 frames/s
 - Displaying with 50 frames/s
- Limited spatial resolution
 - Determines resolution & distance to display

The Human Visual System (HVS) (2)

- Sensor delivers RGB data
- Transformation RGB → YUV / YCbCr color space

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168 & -0.331 & 0.5 \\ 0.5 & -0.418 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

- Signal changes on the luminance channel (Y) result in zero activity of the chromatic channels of the HVS

Color Conversion RGB \rightarrow YCbCr



R



G



B



Y



Cb



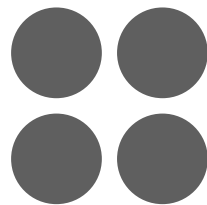
Cr



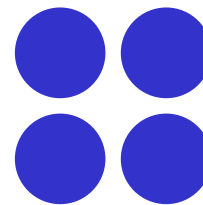
Color Sub-Sampling (1)

- Reduced spatial sensitivity to color information (Chroma)
 - Sub-sampling of color information

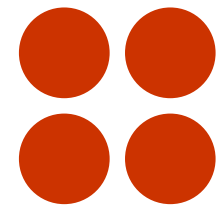
YUV 4:4:4



Y



Cb

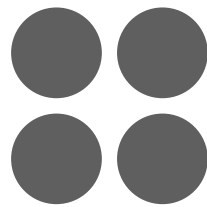


Cr

Color Sub-Sampling (1)

- Reduced spatial sensitivity to color information (Chroma)
 - Sub-sampling of color information

YUV 4:2:2



Y



Cb

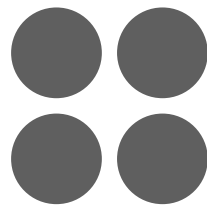


Cr

Color Sub-Sampling (1)

- Reduced spatial sensitivity to color information (Chroma)
 - Sub-sampling of color information

YUV 4:2:0



Y

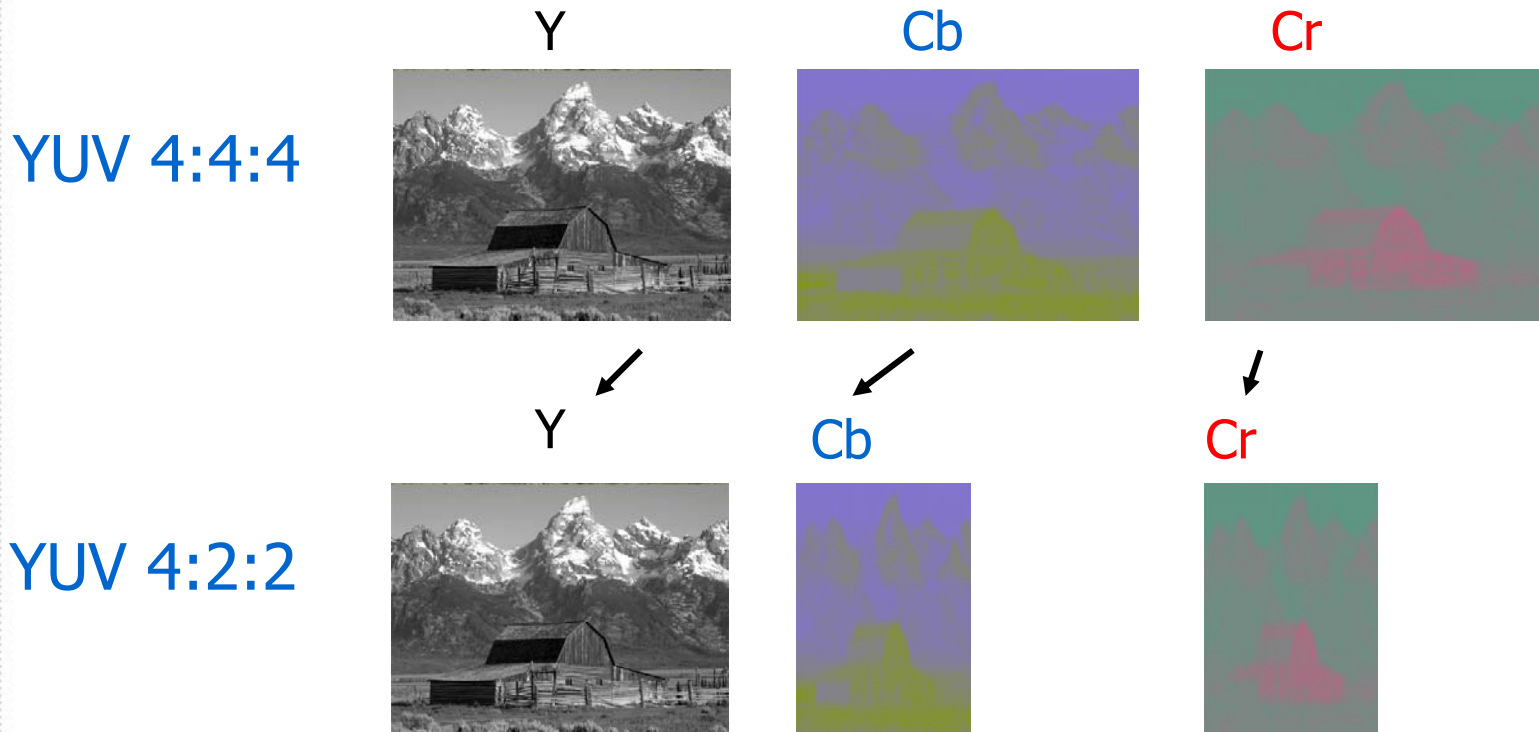


Cb

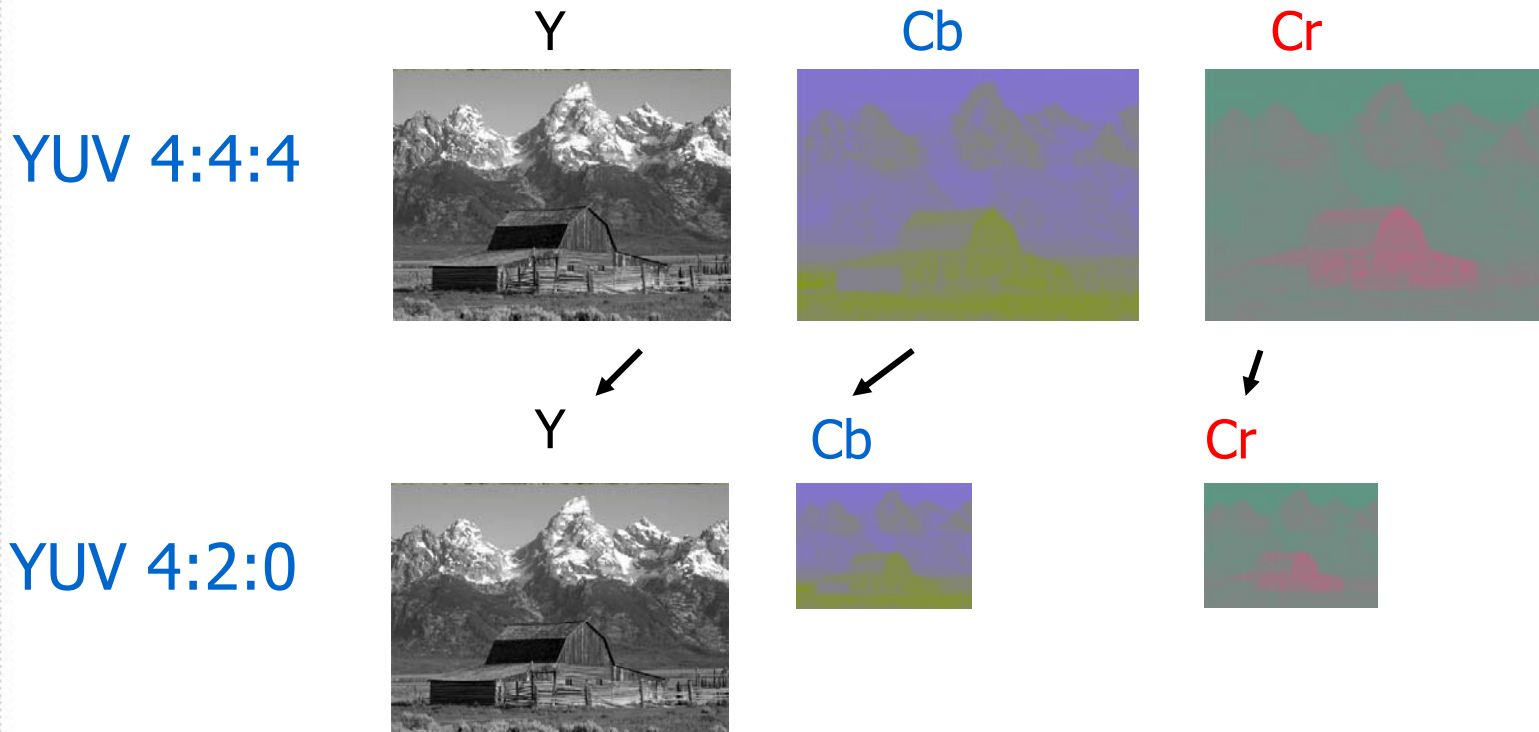


Cr

Color Sub-Sampling (2)

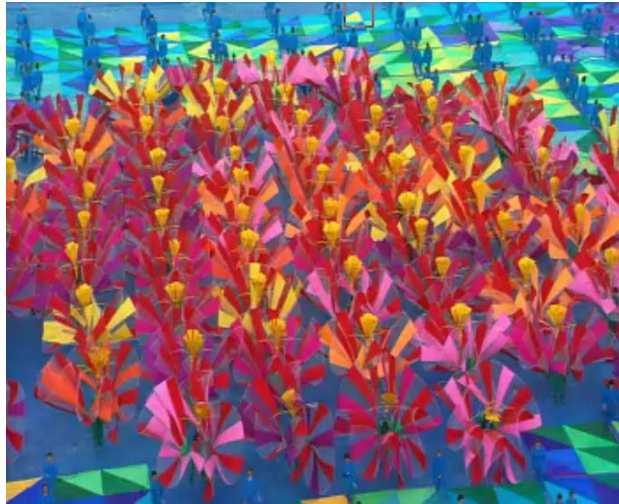


Color Sub-Sampling (2)

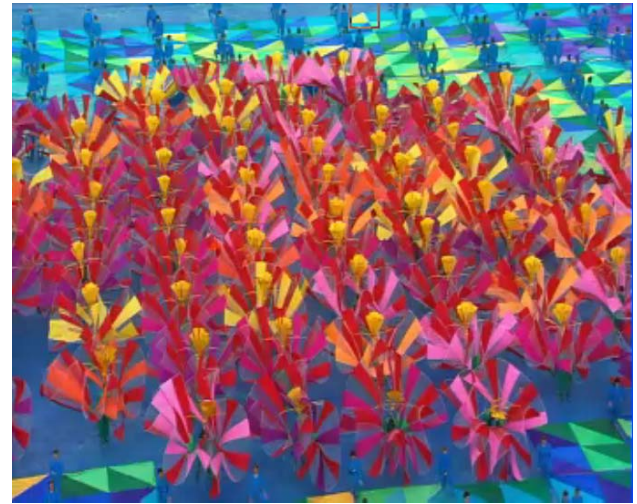


Color Sub-Sampling (3)

YUV 4:4:4

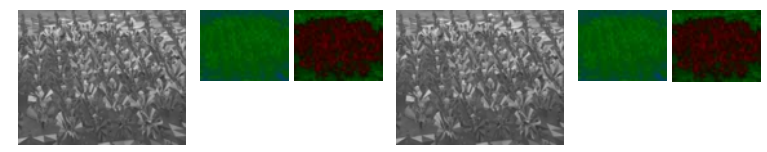


YUV 4:2:0

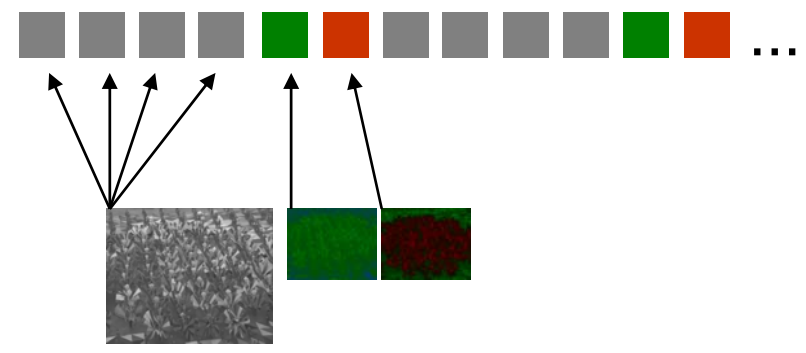


Raw YUV420 Files

Planar



Interleaved



The Digital Age

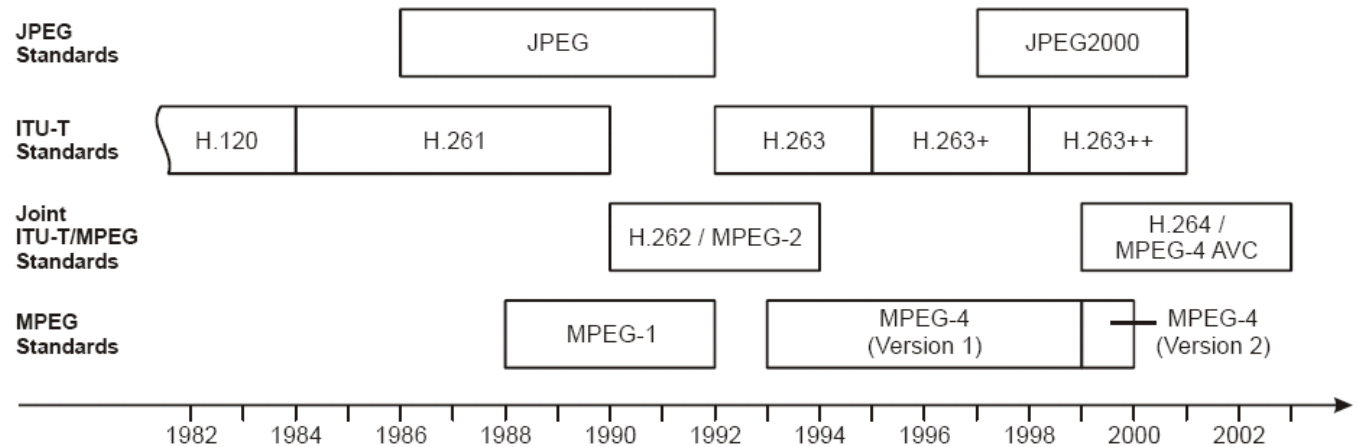
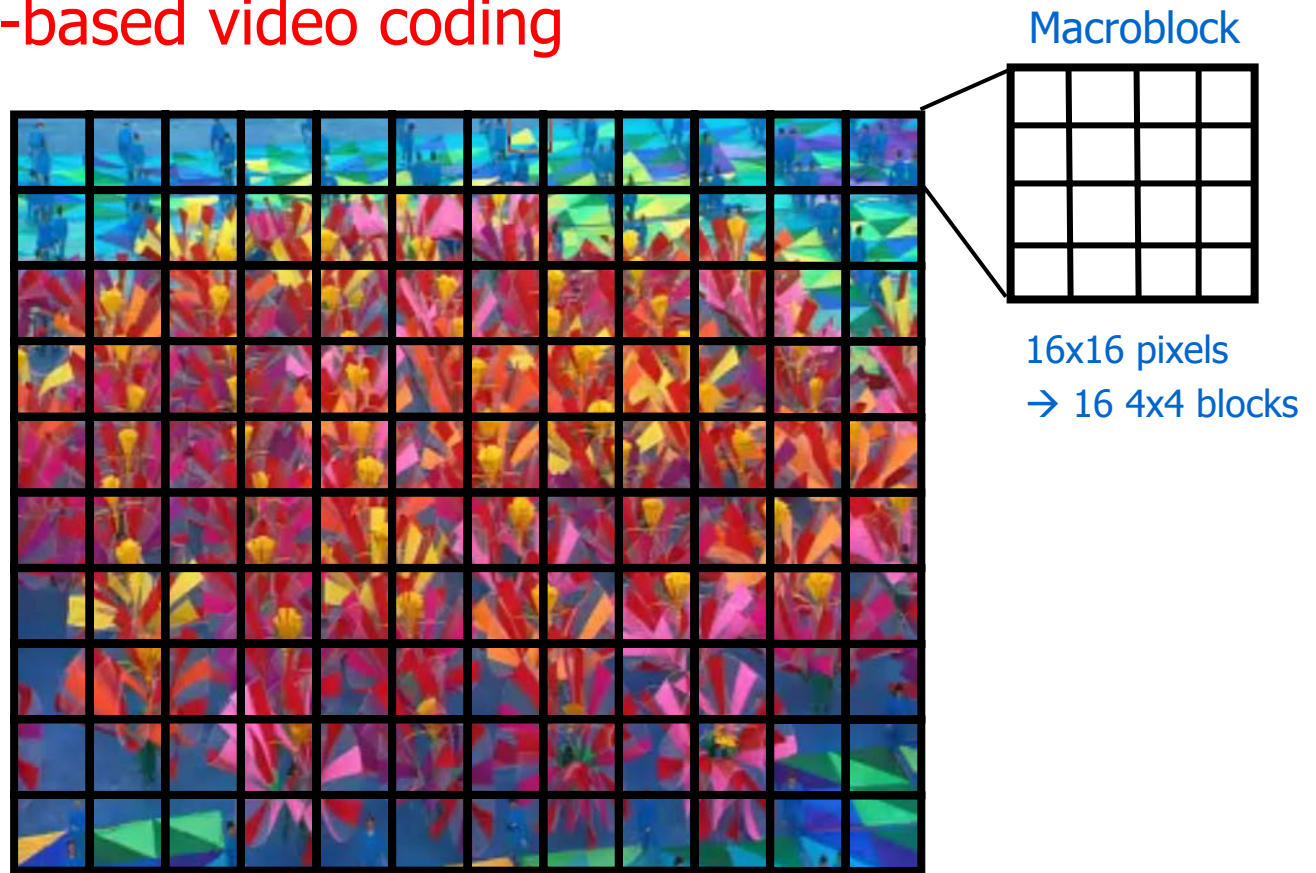
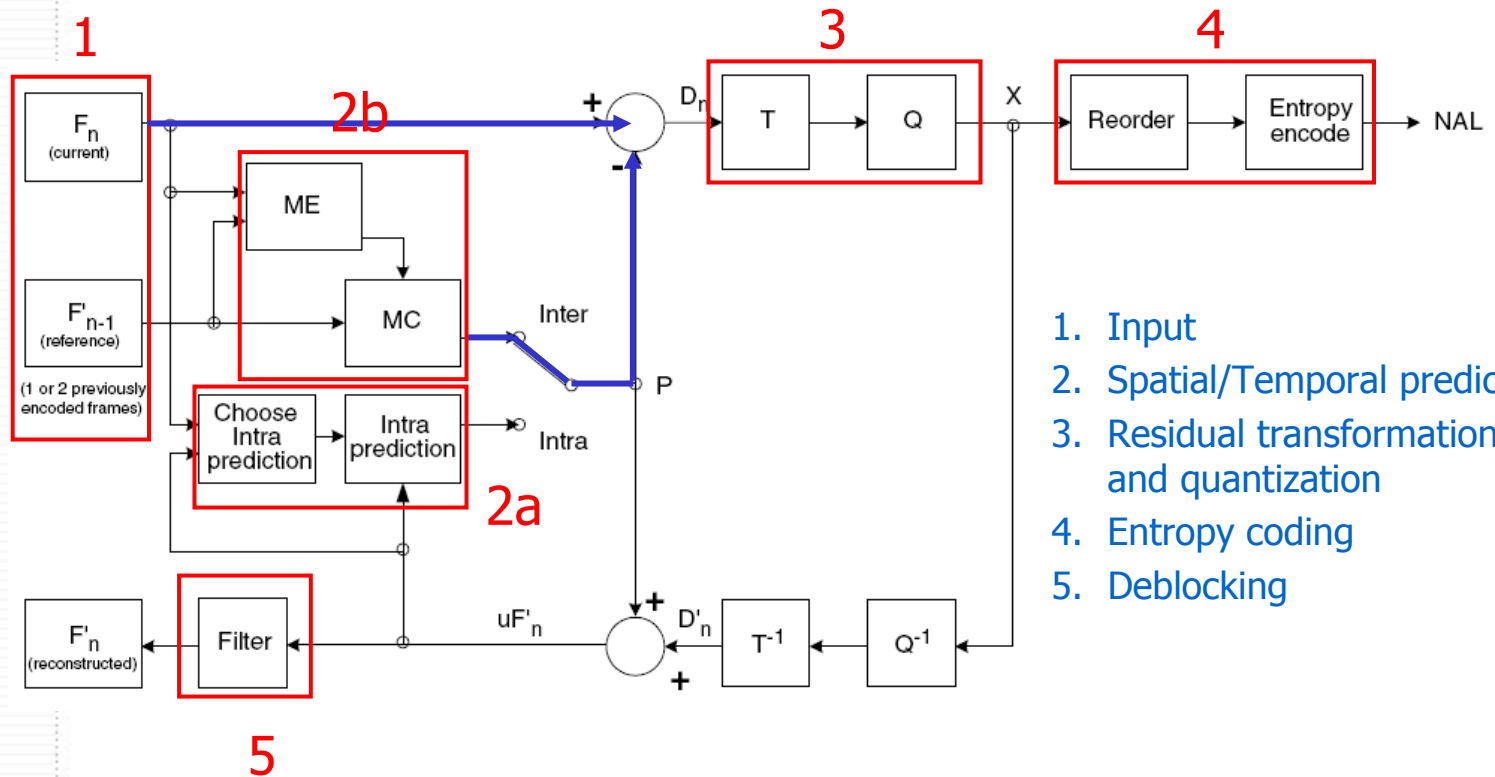


Figure 2.1: Historical development of video standards

Block-based video coding



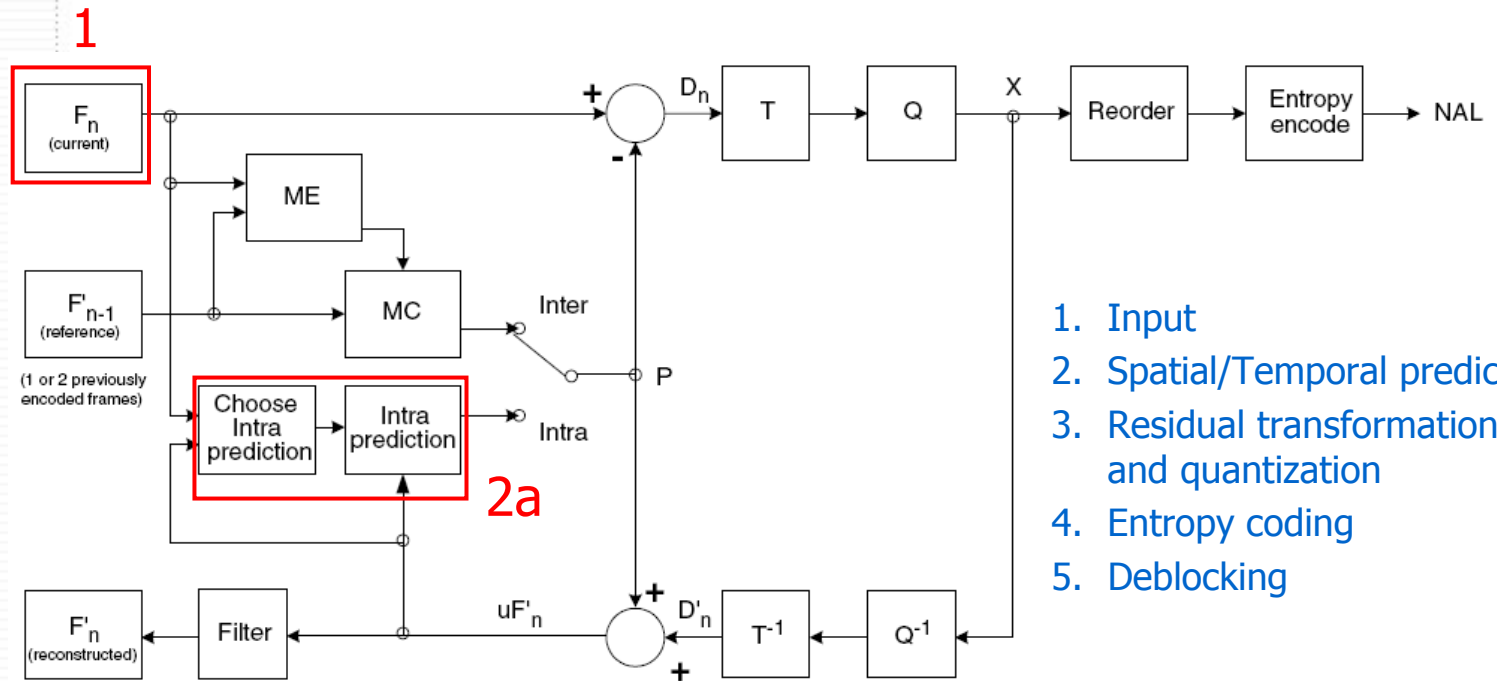
Block-based video coding (H.264 Encoder)



1. Input
2. Spatial/Temporal prediction
3. Residual transformation and quantization
4. Entropy coding
5. Deblocking

→ MPEG-2, MPEG-4 SP/ASP, VC-1, AVS, DivX, etc.

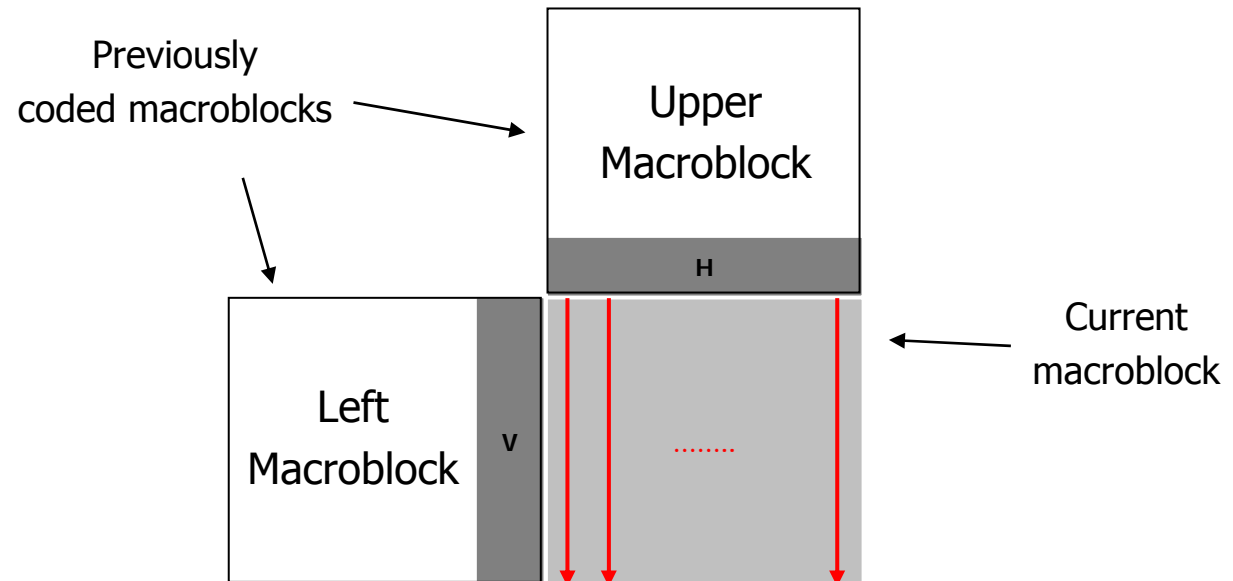
Spatial prediction (Intra Prediction)



1. Input
2. Spatial/Temporal prediction
3. Residual transformation and quantization
4. Entropy coding
5. Deblocking

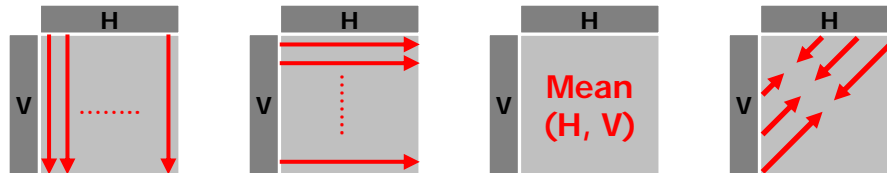
H.264 Intra prediction (1)

- Predict current macroblock's pixels from neighbouring macroblocks

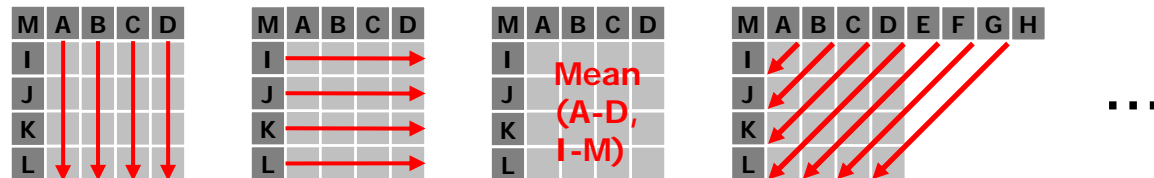


H.264 Intra prediction (2)

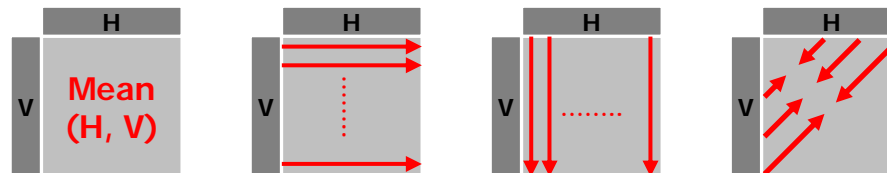
- Different modes for luma and chroma
- Luma 16x16 → 4 modes



- Luma 4x4 → 9 modes



- Chroma: 8x8 Blocks → 4 Modes



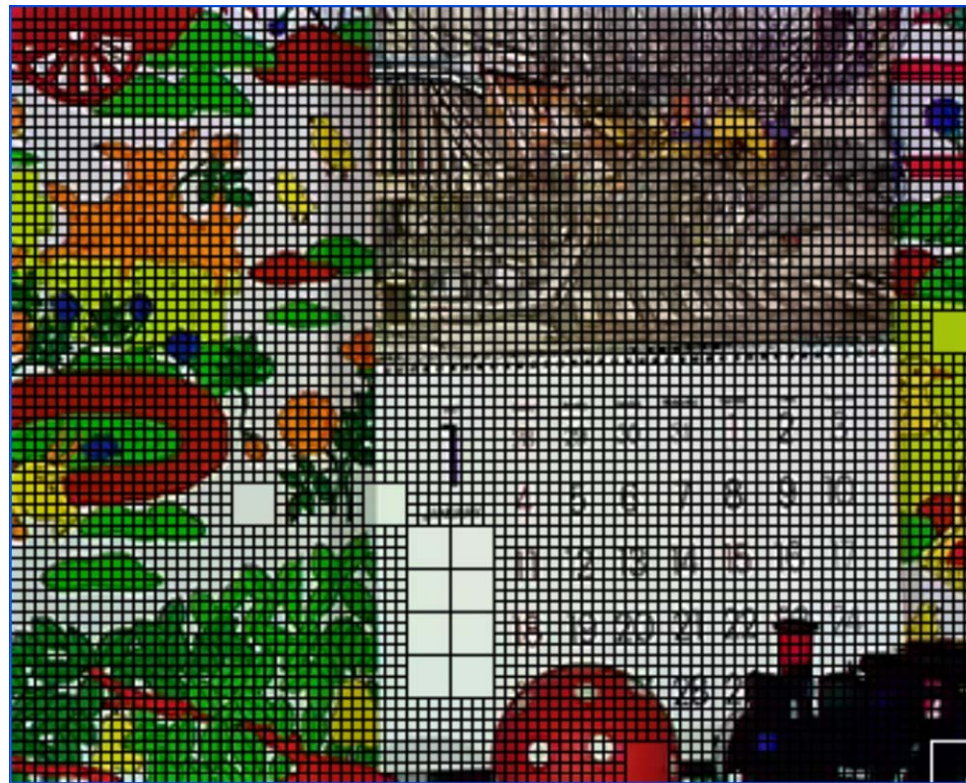
H.264 Intra prediction (3)

- Large intra block size
 - Only few block prediction modes have to be stored
 - Good for untextured regions
(Here coarse prediction achieves good results)
- Smaller intra block size
 - Prediction mode for each 4x4 block required!
 - Better prediction for fine textured structures

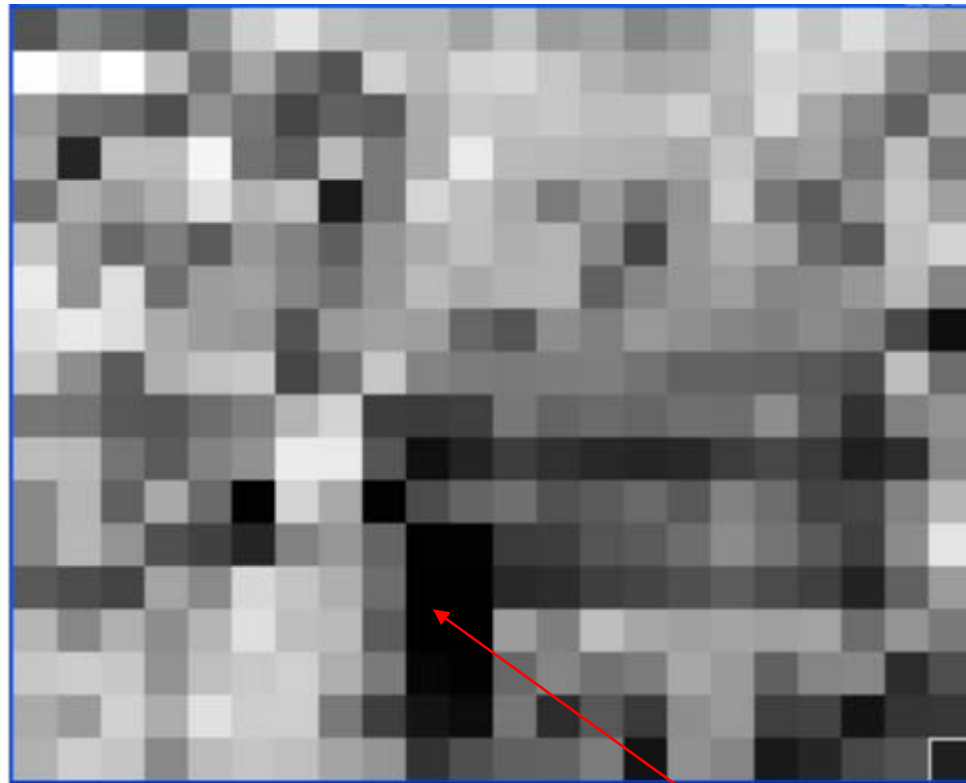
Intra modes / Bitrate (1)



Intra modes / Bitrate (2)

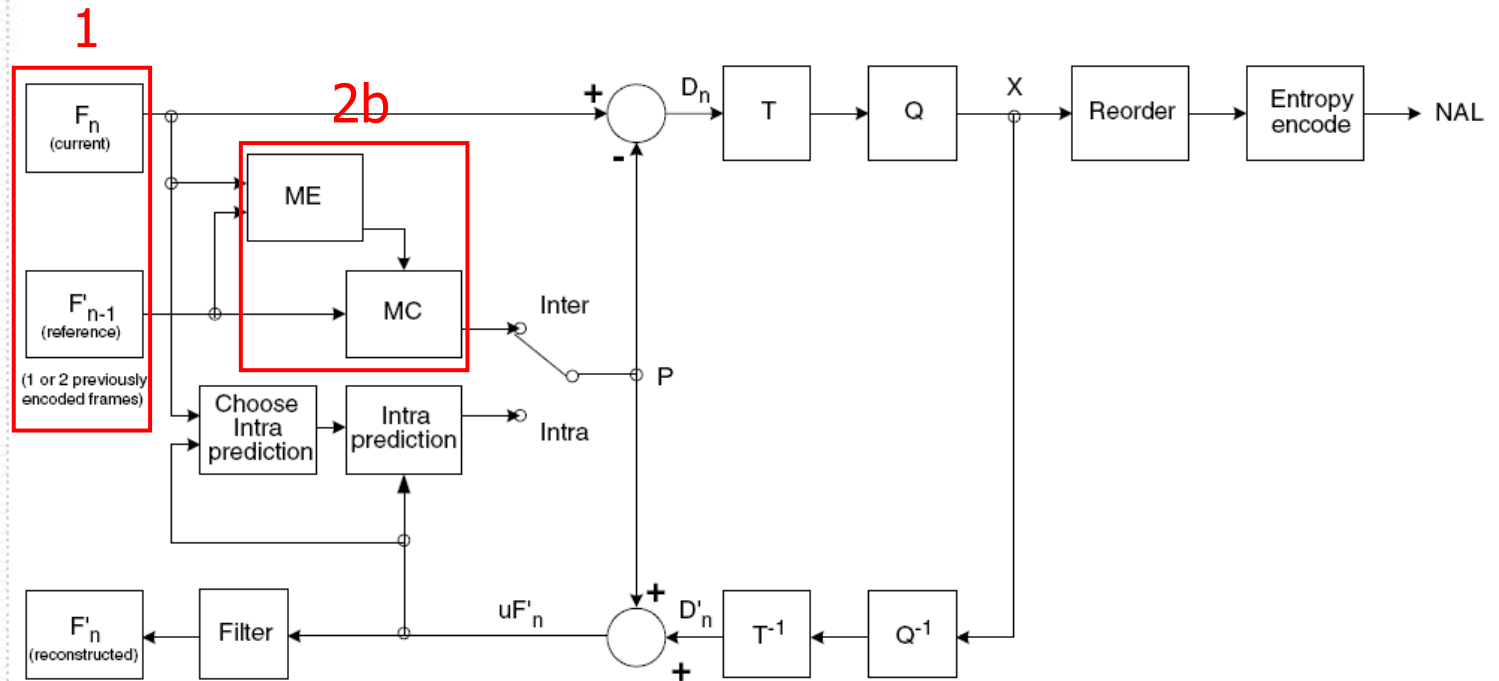


Intra modes / Bitrate (3)



Few bits for coding macroblock

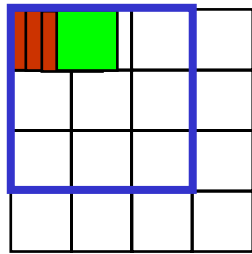
Temporal Prediction (Inter-prediction)



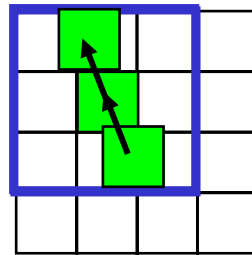
Exploit similarities between frames

Motion Estimation (1)

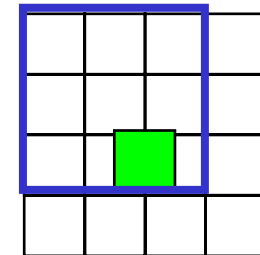
- Search each macroblock in previous frame
 → Search range where we look for most similar MB



Previously coded
frame x



Current frame



Previously coded
frame y

→ Motion Vector (MV)

- Various correlation metrics (e.g. SAD, SSD)
- Computationally demanding (up to 80% of processing time)
- Uni-directional prediction (P-frames)
- Bi-directional prediction (B frames)

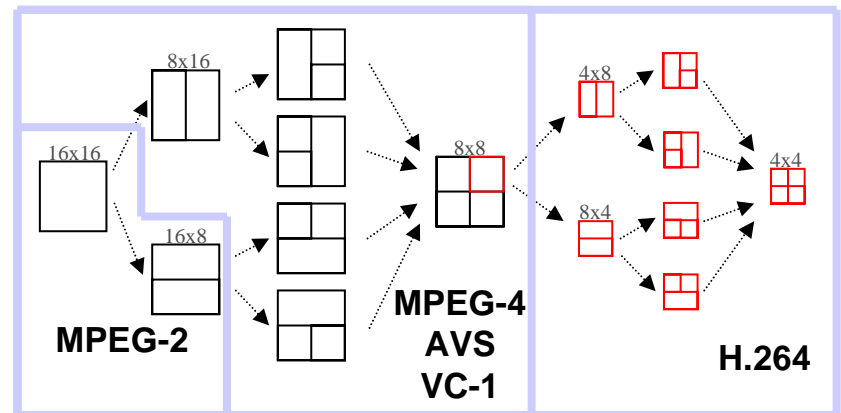
Motion Estimation (2)

- Motion typically at sub-pixel level
- Determine MV with sub-pixel precision
 1. Up-scaling of images
 2. Motion search in up-scaled image
 - MV at sub-pixel accuracy

Features	MPEG-2	MPEG-4 ASP	H.264
MV resolution	1/2 pixel	1/4 pixel	1/4 pixel

Motion Estimation (3)

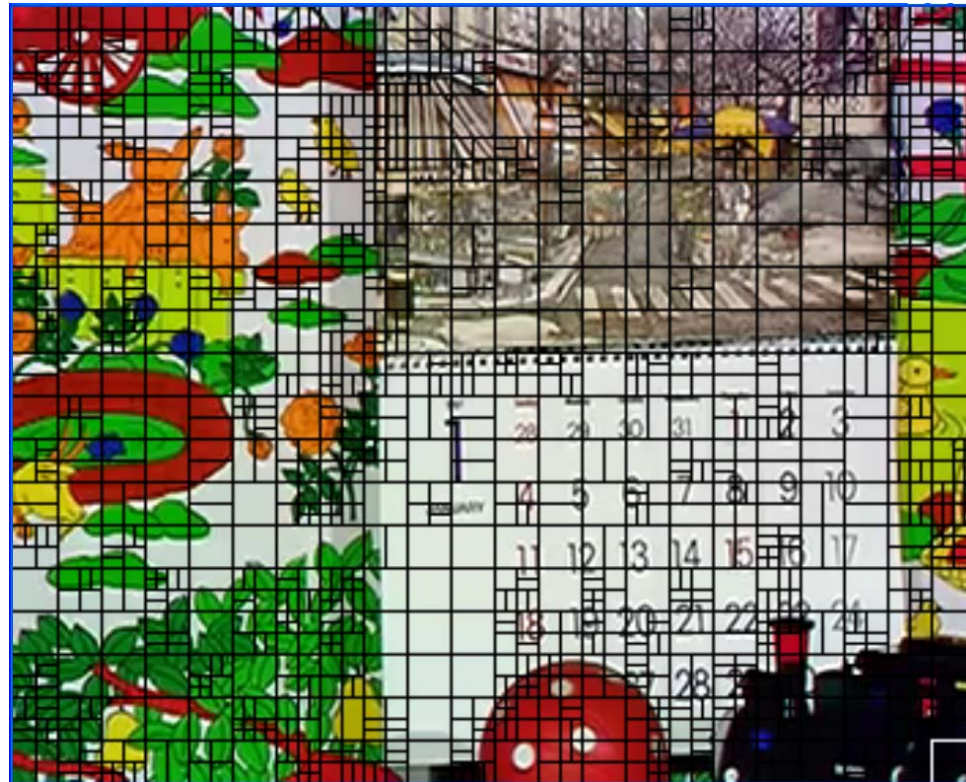
- Fine motion is addressed by smaller block sizes
- Variable block sizes



- Each sub-block has an individual MV

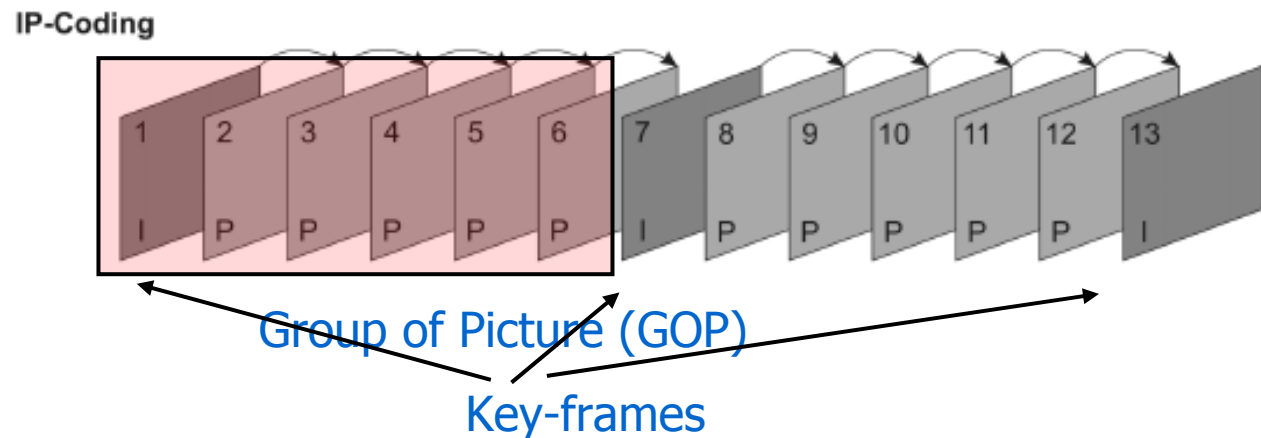
Features	MPEG-2	MPEG-4 ASP	H.264
Vector block size	16x16, 16x8	16x16, 8x8	16x16, 16x8, 8x16, 8x8, 8x4, 4x8, 4x4

Temporal prediction (Partition size)



Video structure

- Frame prediction determines structure

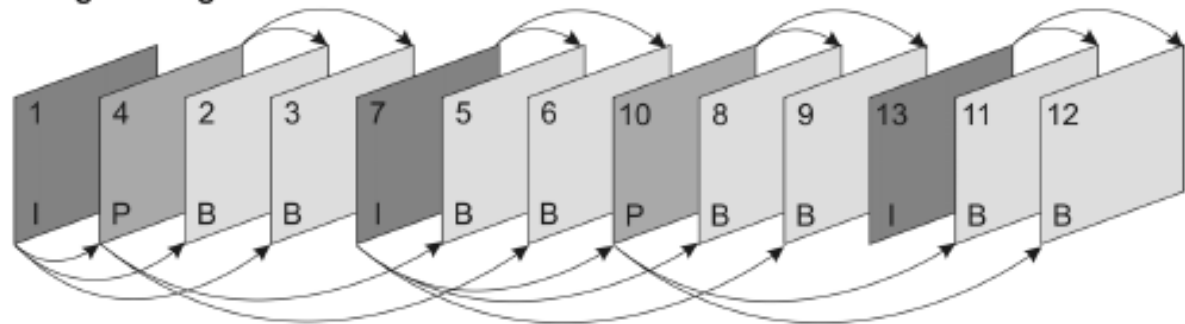


- Keyframes allow start of decoding at arbitrary positions in the video (e.g. for channel switching)
- #Keyframes dependent on the application
Example: Television broadcasting typically with 8-15 frames/GOP

Video structure

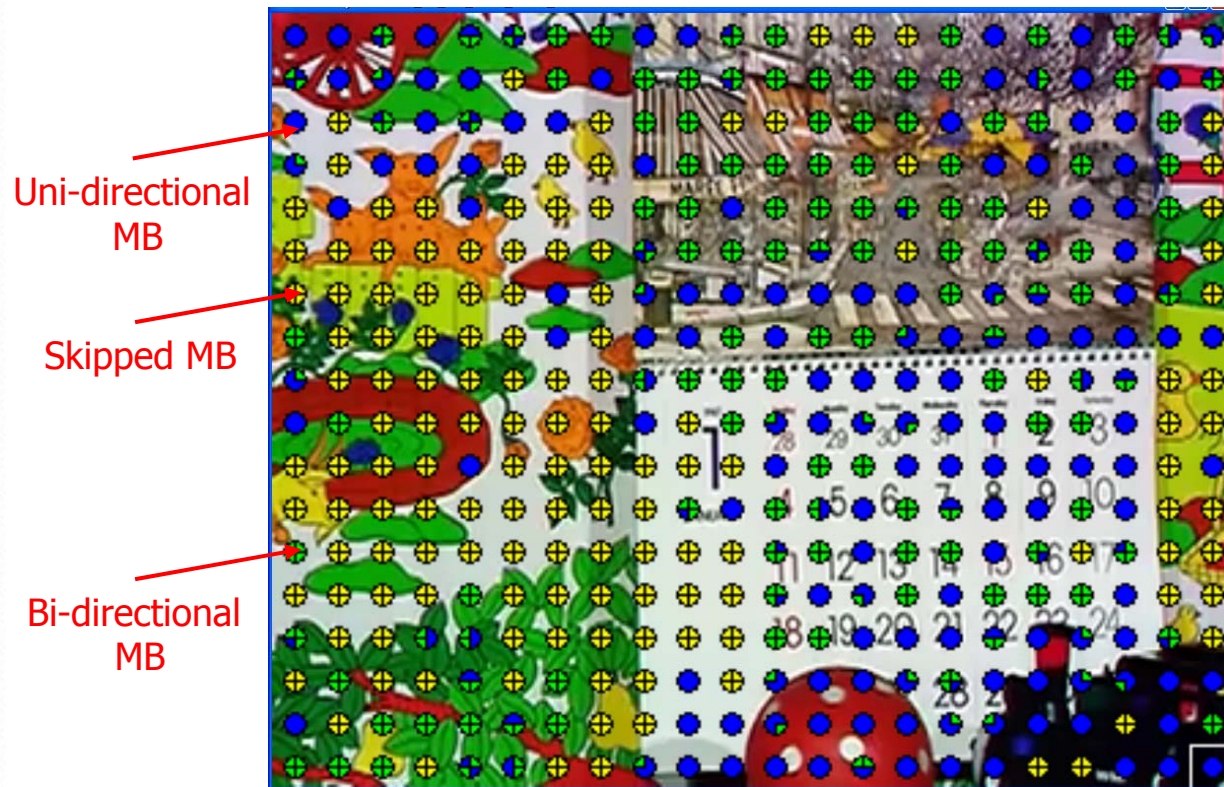
- Bi-directional prediction (B-frames)

IBBP-Coding: Coding Order



Features	MPEG-2	MPEG-4 ASP	H.264
P-frames	1	1-4	1-5
B-frames	1 / direction	Not in SP	multiple / direction

Temporal prediction (Coding mode)



B-frame

H.264 Intra vs. Inter prediction (1)

Intra

- Spatial correlation
 - Pixel-based
 - Simple pixel propagation
- Saves around 6-9% bits

Inter

- Temporal correlation
 - Block-based
 - Structure and intensity propagation
- Saves around 50% bits

H.264 Intra vs. Inter prediction (2)

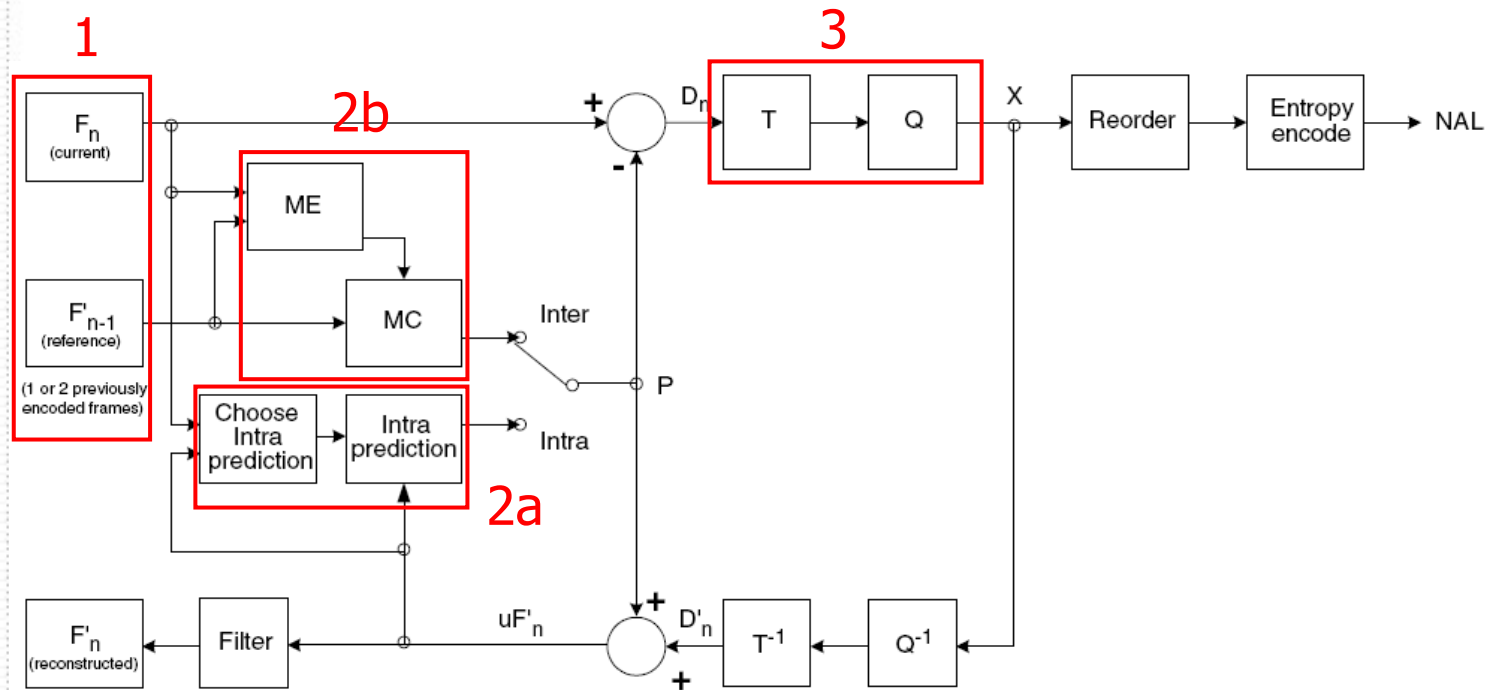
Intra

- Dependency on spatial neighbour MBs
- Small feed-back loop
- Unfiltered pixels as reference

Inter

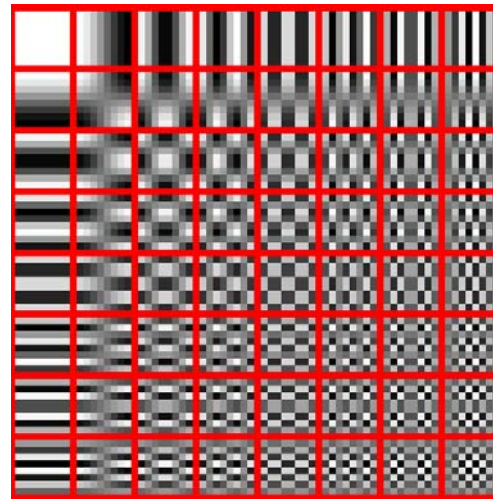
- Dependencies on other frames
- Large feedback loop
- Filtered pixel as references (In-loop filtering)

Transformation / Quantization



Discret Cosine Transform (DCT)

- Eye is less sensitive to high frequencies
- Transform residuals to frequency domain
- Express data points as combination of 2D cosine functions at different frequencies



2D cosine functions

Discret Cosine Transform (2)

- 4x4 DCT Transformation

$$Y = A X A^T = \begin{pmatrix} \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} & X & \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \end{pmatrix}$$

$$a = 1/2 \quad b = \sqrt{1/2} * \cos\left(\frac{\Pi}{8}\right) \quad c = \sqrt{1/2} * \cos\left(\frac{3\Pi}{8}\right)$$

Problem of DCT:

- Transformation accuracy (numerical issues)
- Previous standards specify precision requirements

Discret Cosine Transform (3)

- 4x4 DCT Transformation

$$X = \begin{bmatrix} 136 & 142 & 148 & 161 \\ 138 & 145 & 149 & 164 \\ 131 & 143 & 150 & 158 \\ 135 & 139 & 149 & 162 \end{bmatrix} \xrightarrow{\text{DCT}} Y = \begin{bmatrix} 587.5 & -37.95 & 5 & -5.39 \\ 2.55 & 1.96 & 0.97 & -1.81 \\ -1.50 & -0.35 & 3 & 1.77 \\ -4.30 & -0.31 & -4.19 & 0.54 \end{bmatrix}$$

Discret Cosine Transform (4)

$$Y = AXA^T = \begin{pmatrix} \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} & \begin{bmatrix} X \end{bmatrix} & \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix} \end{pmatrix}$$

$$a = 1/2 \quad b = \sqrt{1/2} * \cos\left(\frac{\Pi}{8}\right) \quad c = \sqrt{1/2} * \cos\left(\frac{3\Pi}{8}\right)$$

$$Y = CXC^T \otimes E = \begin{pmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} & \begin{bmatrix} X \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \end{pmatrix} \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}$$

$$d = c/b$$

H.264 Integer Transform

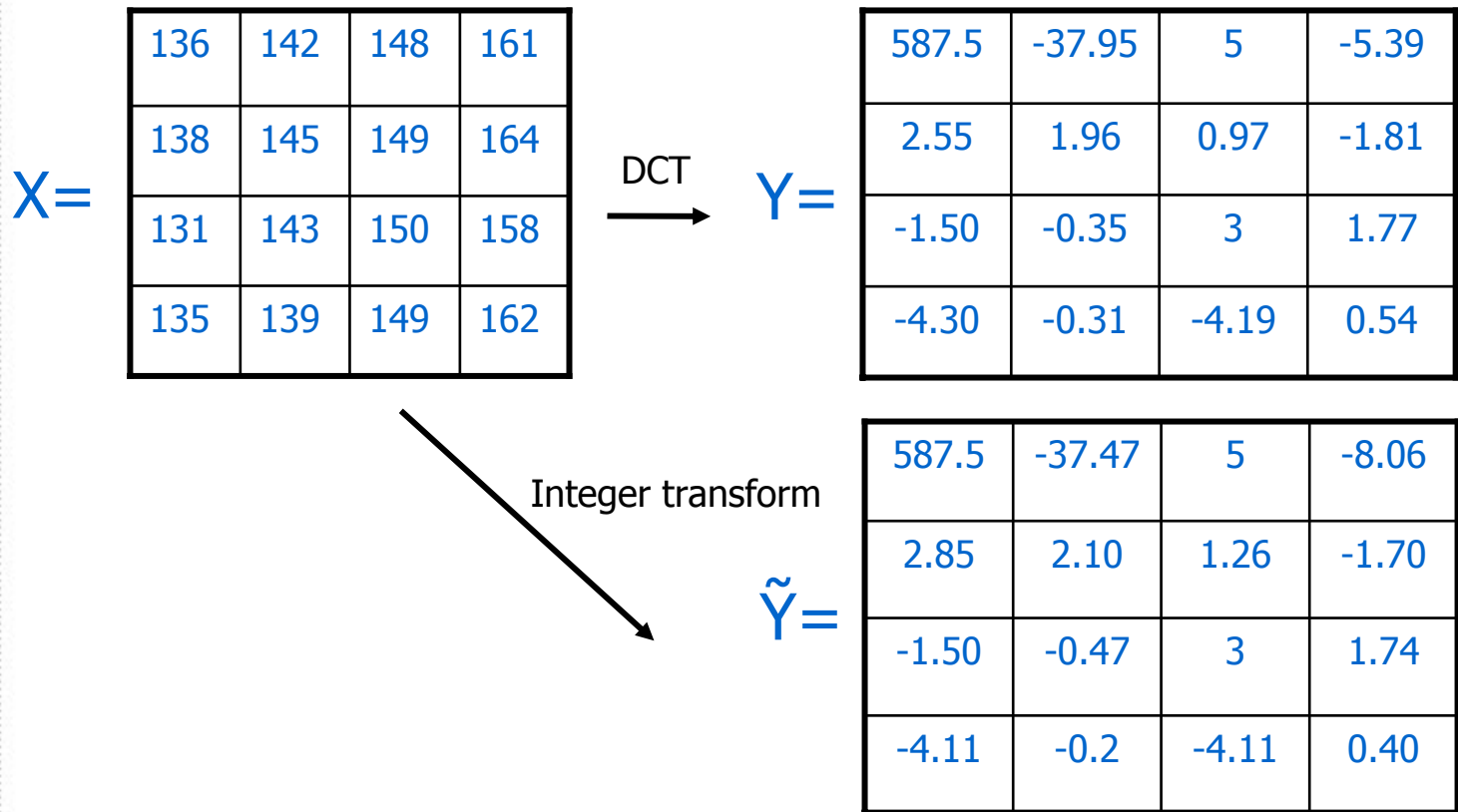
- An approximation to the DCT

$$Y = H X H^T \otimes E = \underbrace{\begin{pmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} & \begin{bmatrix} X \\ \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \end{bmatrix}}_{\text{Integer transform}} \otimes \underbrace{\begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix}}_{\text{Post-scaling factor (PF)}}$$

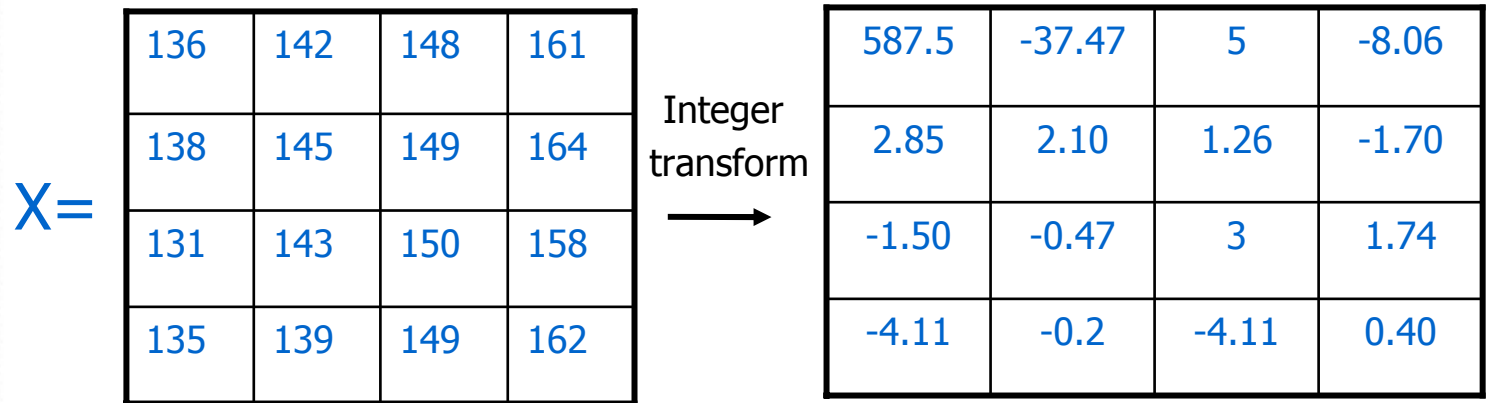
$$a = 1/2 \quad b = \sqrt{2/5}$$

- Exact inverse transform is possible (Accuracy, Implementation independent)
- Only adds and shifts
- 16-bit calculations
- 4x4 Blocks → smaller blocks result in fewer noise around the edges (ringing)

H.264 Integer Transform



Quantization

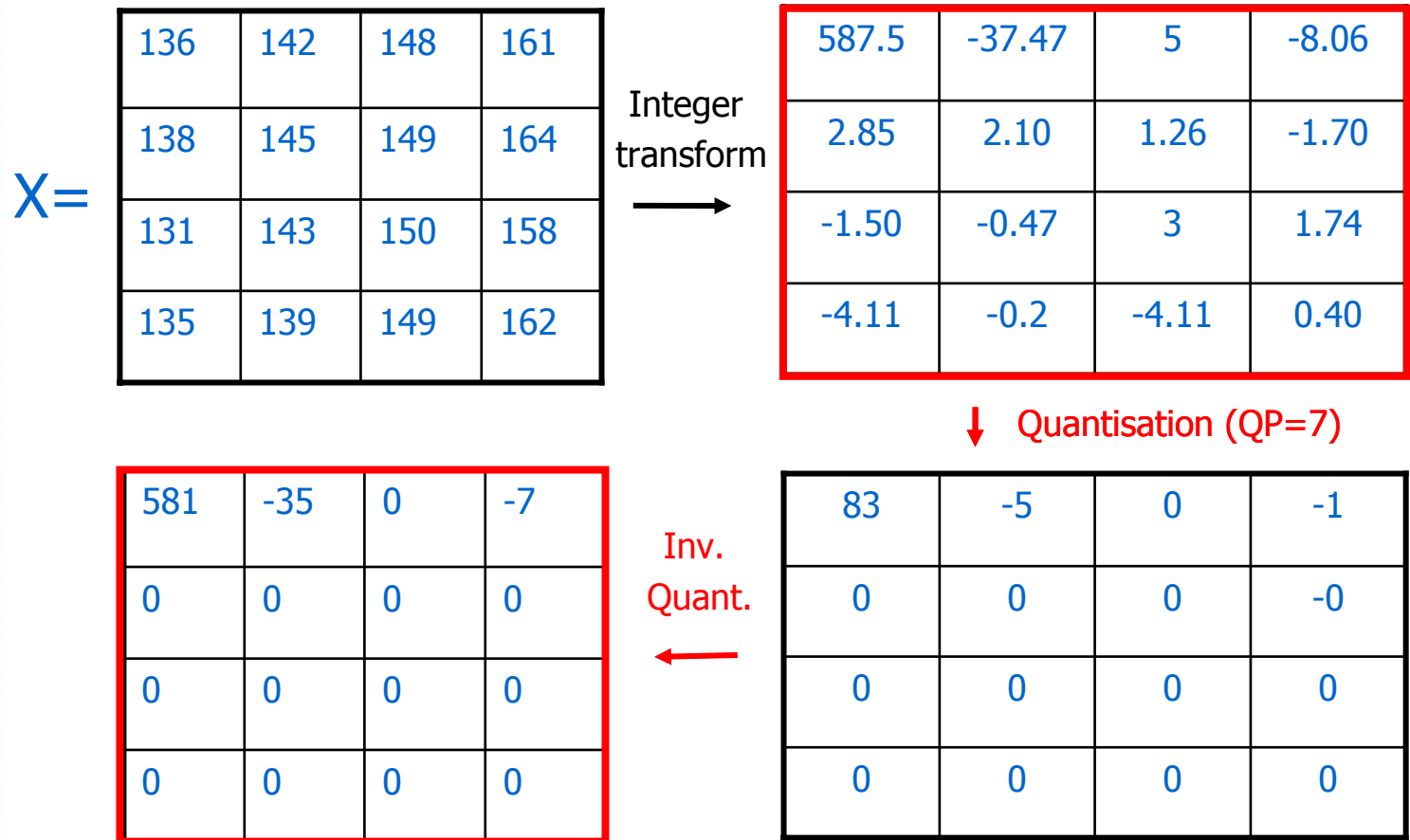


↓ Quantisation (QP=7)

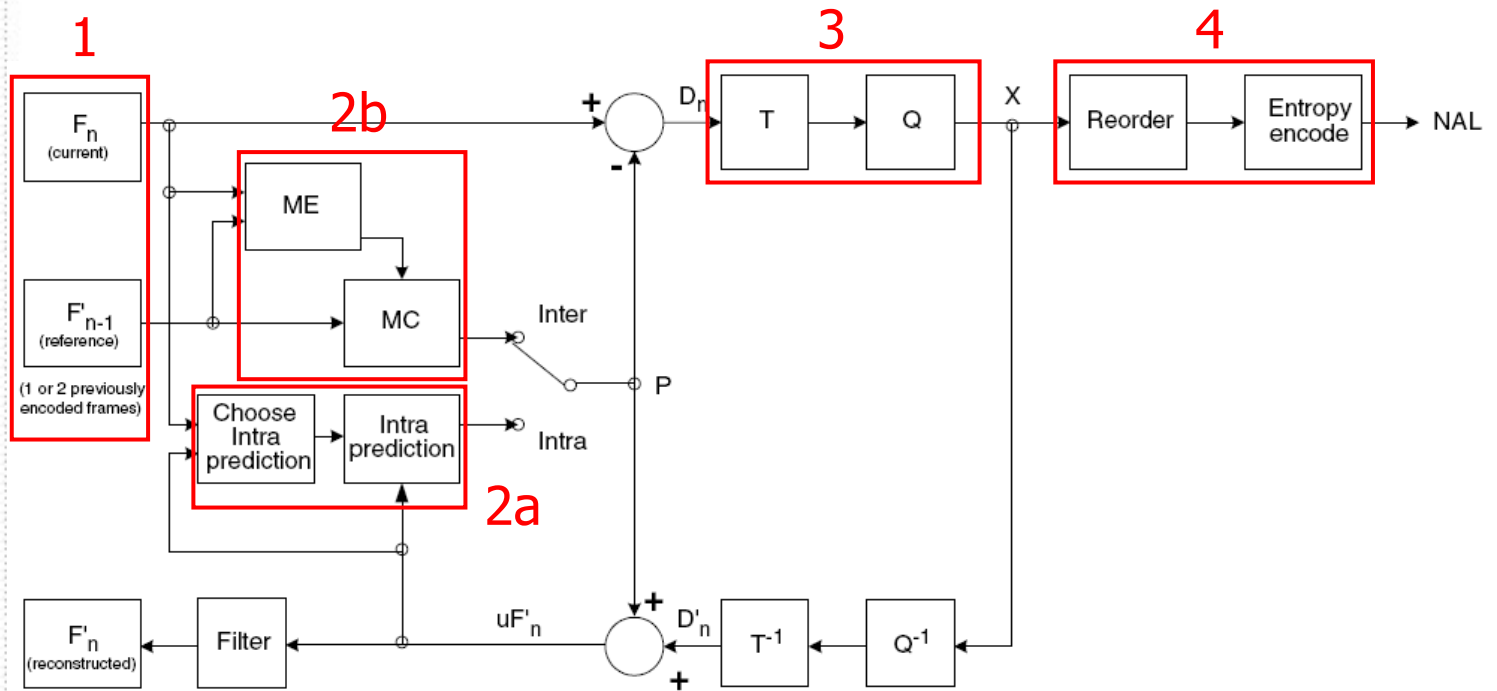
83	-5	0	-1
0	0	0	-0
0	0	0	0
0	0	0	0

- Here information loss takes place!
- Rate control at this point
 - Quantization Parameter (QP)

Quantization

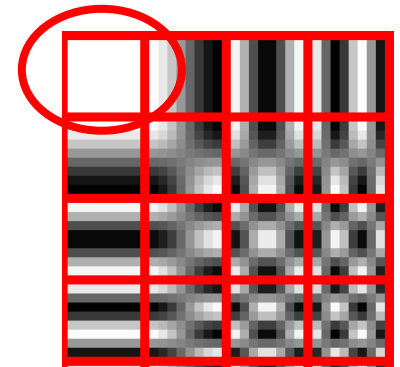


Reordering / Entropy coding



Reordering (Zig-Zag Scan)

83	-5	0	-1
0	0	0	0
0	0	0	0
0	0	0	0



- Most energy represented by low frequencies
 - Large number of zeros
 - Zig-Zag Ordering

Entropy coding

- Remove statistical redundancy
- Variable length coding (VLC)
- Arithmetic Coding (AC)
- Context-Adaptive coding

Huffman Coding

1. Sequence of symbols

BAACABAAAD A=00, B=01, C=10, D=11

fixed-length coding:

4 different symbols \rightarrow 2 Bits/Symbol \rightarrow 10 x 2 (=20 Bits)

2. Compute probability of each symbol

$p(A) = 0.6$, $p(B)=0.2$, $p(C)=0.1$, $p(D)=0.1$

3. Assign code words according to probability (Coding table)

A \rightarrow 1, **B \rightarrow 01**, **C \rightarrow 001**, **D \rightarrow 000** (higher prob. \rightarrow shorter words)

4. Code symbols with this code table

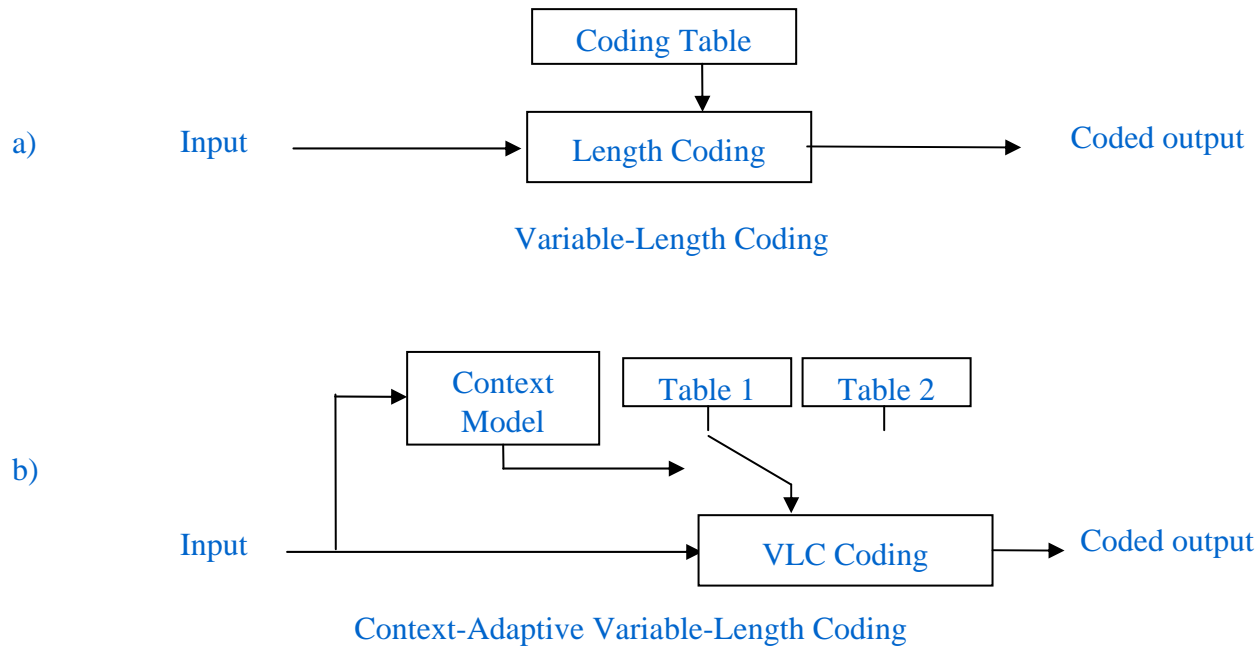
BAACABAAAD \rightarrow 01 1 1 001 1 01 1 1 1 000 (=16 Bits)

\rightarrow Probability density function stream dependent!

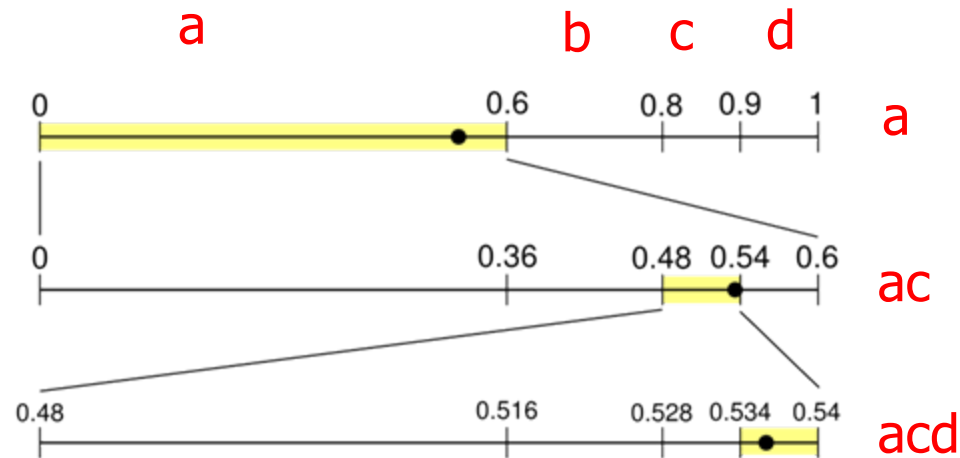
Context-Adaptiv VLC (1)

- Assumption:
Residuals of spatially close MBs have similar values!
- Exploit this correlation!
- Multiple coding tables
- Switch tables according to neighbouring MBs

Context-Adaptiv VLC (2)



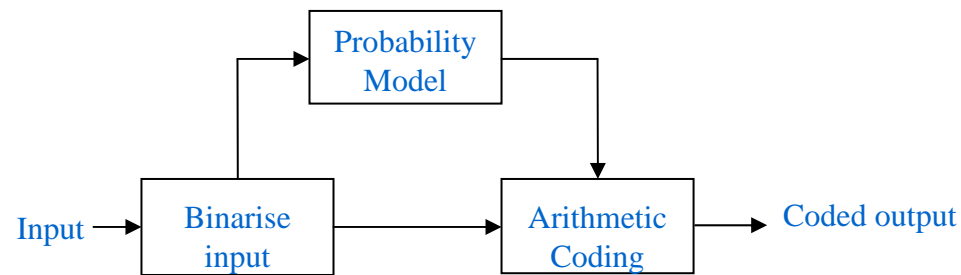
Arithmetic Coding (AC)



- Any number in the range $[0.534, 0.54]$ represents the symbol sequence **acd**.
- Use the number which can be coded most efficiently in binary representation.
- The bits allocated for each symbol can be non-integer!
- For a known pdf, AC always equal or better than Huffman coding

Binary-Arithmetic Coding

- Arithmetic coding using 2 probability states
- Binarization of non-binary valued symbols
 - Syntax elements (SEs): MV, residuals, coding mode, etc.
 - SE \rightarrow bins
 - Different SE binarizations
- Context-Adaptive BAC
 - Context probability model selection based on statistics of the recently coded symbols

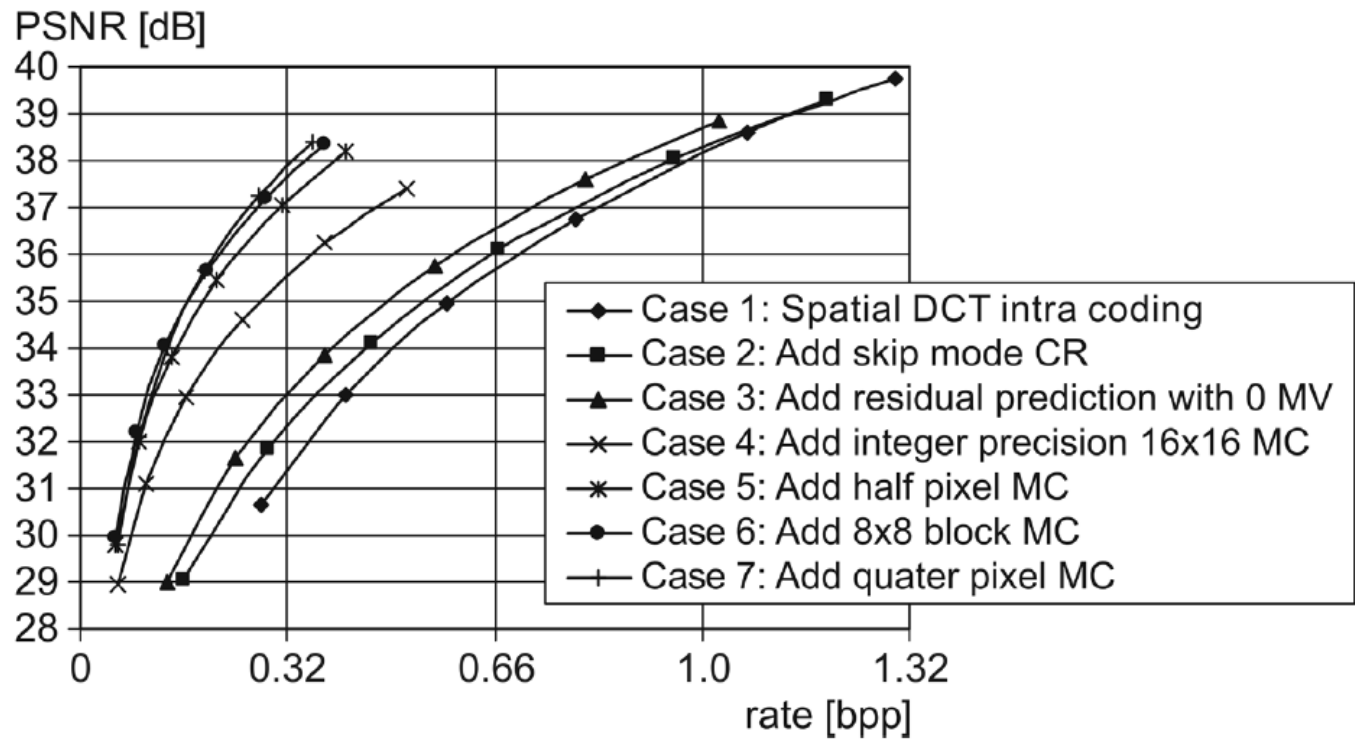


Context-Adaptive Binary Arithmetic Coding

Comparison CAVLC / CABAC

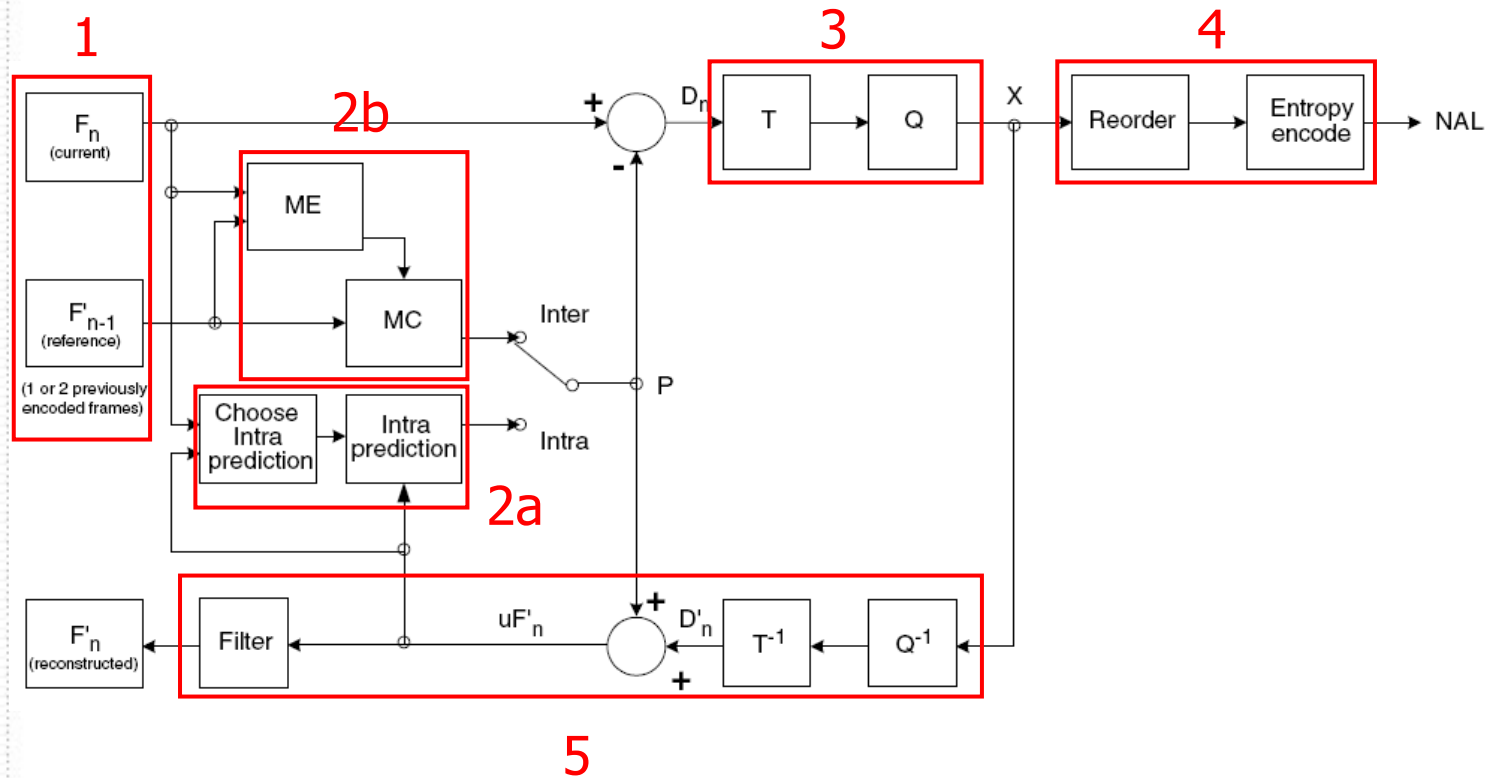
- CAVLC
 - Saves 5-8% bits
- CABAC
 - Saves 5-15% bits over CAVLC
 - Integer and multiplication free CABAC in H.264!
 - Higher computational complexity than CAVLC
 - CAVLC for platforms with lower computational power

Quality vs. Bitrate

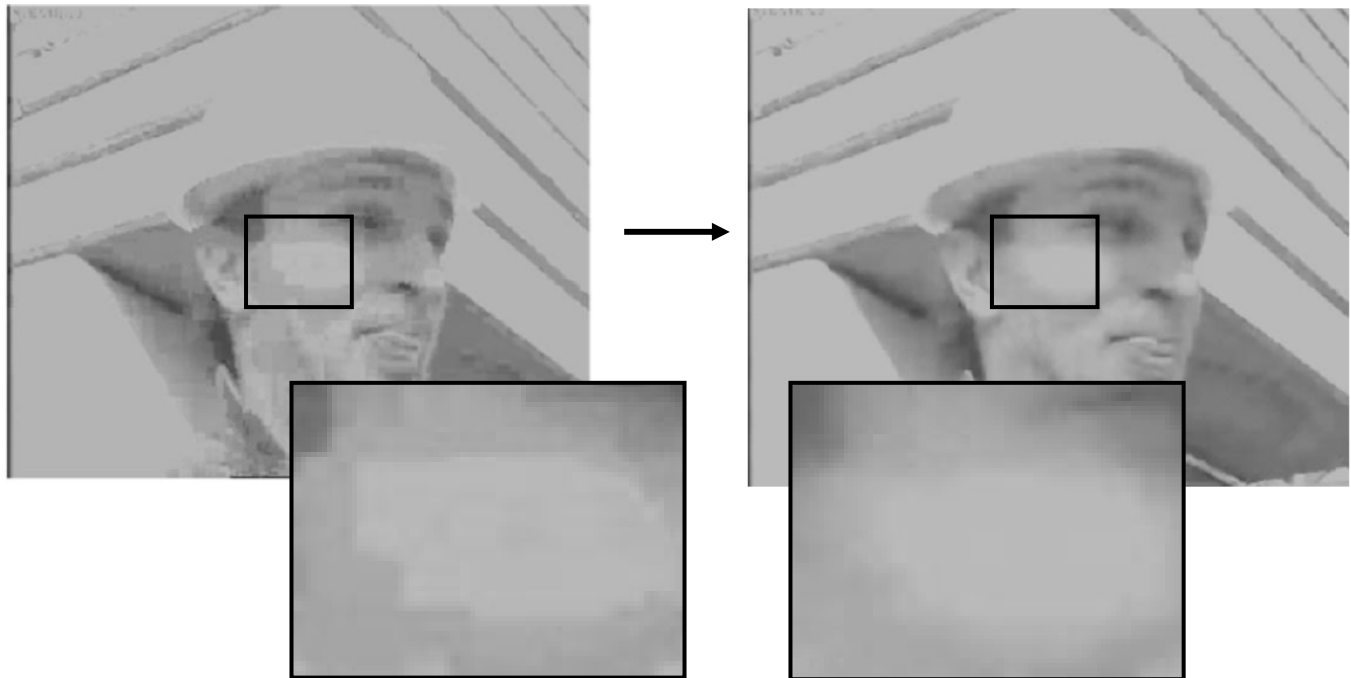


(Sullivan, Wiegand, 2003)

Deblocking



Deblocking Filter (1)

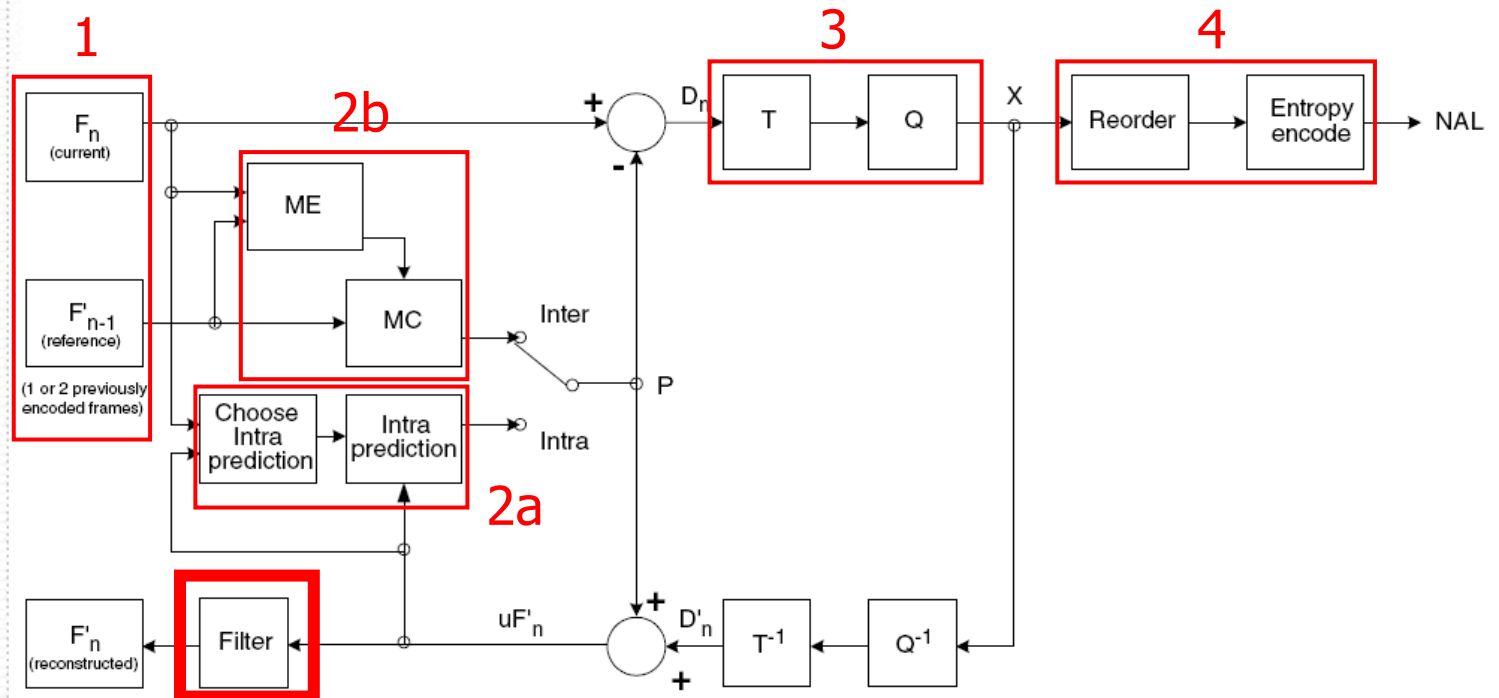


Deblocking Filter (2)

- Removal of blocking artifacts
- Artifacts caused by block-based coding
- Post-processing vs. In-loop deblocking
- In-loop deblocking must be covered by standard

Features	MPEG-2	MPEG-4 ASP	AVS	VC-1 / WM-9	H.264
Deblocking	Post	Post	In-loop	In-loop	In-loop

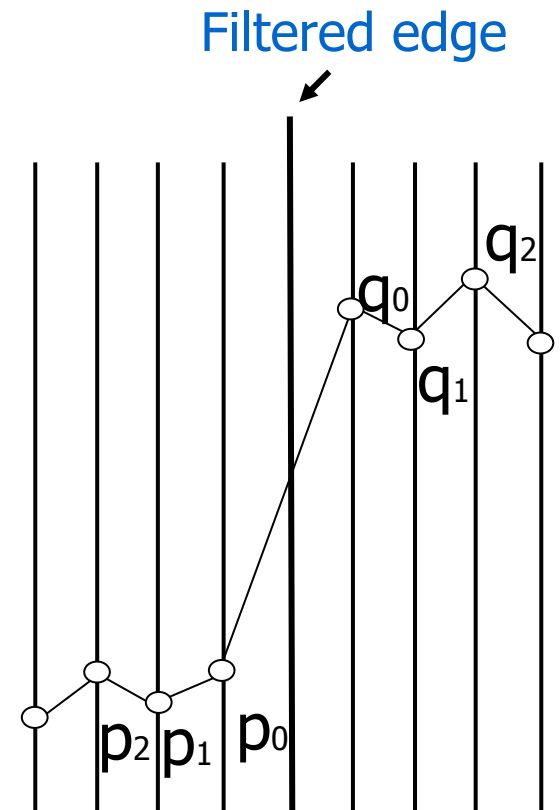
Deblocking Filter (2)



Filtered frames used for prediction

H.264 Deblocking Filter (1)

- Saves about 5-10% bits
- Removing of blocking artifacts
- Filtering on 4x4 blocks
- Heuristic for distinguish between „real“ edges & artifacts



H.264 Deblocking Filter (2)

1. Assign a boundary strength (BS) to every 4x4 block
2. Determining filter operation

Block → No filtering conditions	BS
BS = 0 → No filtering	0
BS = 1 → Slight filtering One of the blocks is intra-coded and the edge is a BS edge	4
BS = 2 → Strong filtering	4
One of the blocks is intra-coded	3
One of the blocks has coded residuals	2
Difference of block motion \geq one luma sample distance	1
Motion compensation from different reference frames	1
Else	0

H.264 Deblocking Filter (2)

- Filtering p_0 and q_0 only if

$$|p_0 - q_0| < \alpha(QP)$$

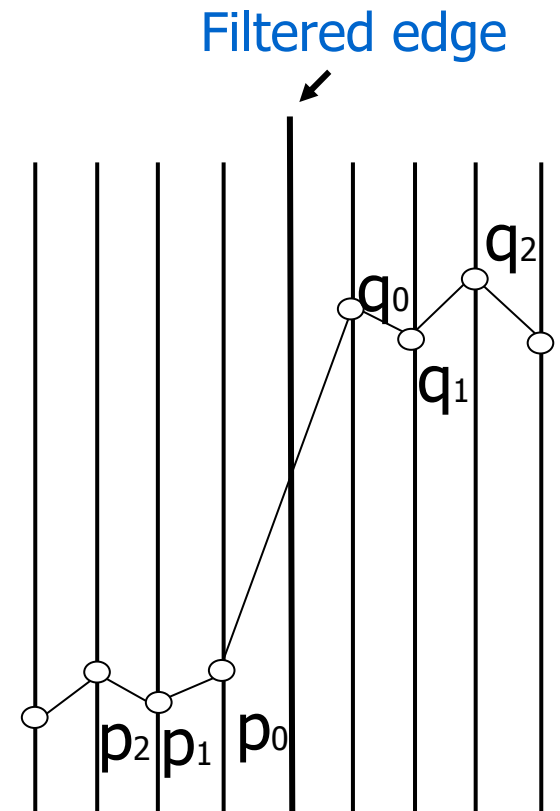
$$|p_1 - p_0| < \beta(QP)$$

$$|q_1 - q_0| < \beta(QP)$$

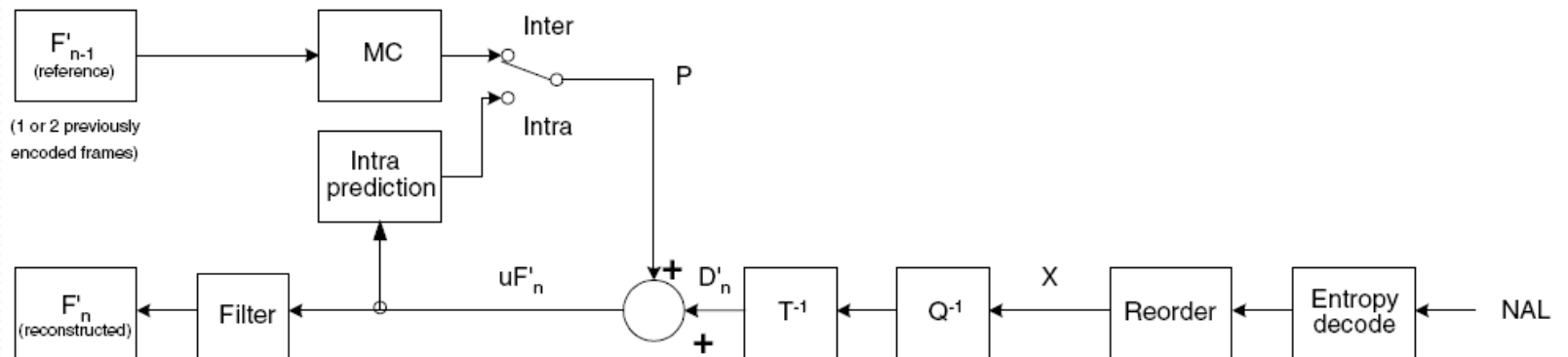
- Filtering of p_1 and q_1 only if

$$|p_2 - p_0| < \beta(QP)$$

$$|q_2 - q_0| < \beta(QP)$$



Block-based video coding (H.264 Decoder)



Key improvements of H.264

- Enhanced Intra-/Inter prediction
- Small block sizes (4x4)
- Exact integer transformation
- Enhanced entropy coding
- Adaptive in-loop deblocking

Resource Scarcity

- Computation power (33 - 300 MHz)
- Local memories / caches (16 - 128 kB)
- External memory accesses
- Bus bandwidth

- Power consumption
- Chip area / cost

Main Applications

- Video conferencing and video telephony
- Video broadcasting
 - Digital video broadcast (DVB)
 - Digital media broadcast (DMB)
- Video storage and retrieval
- Streaming

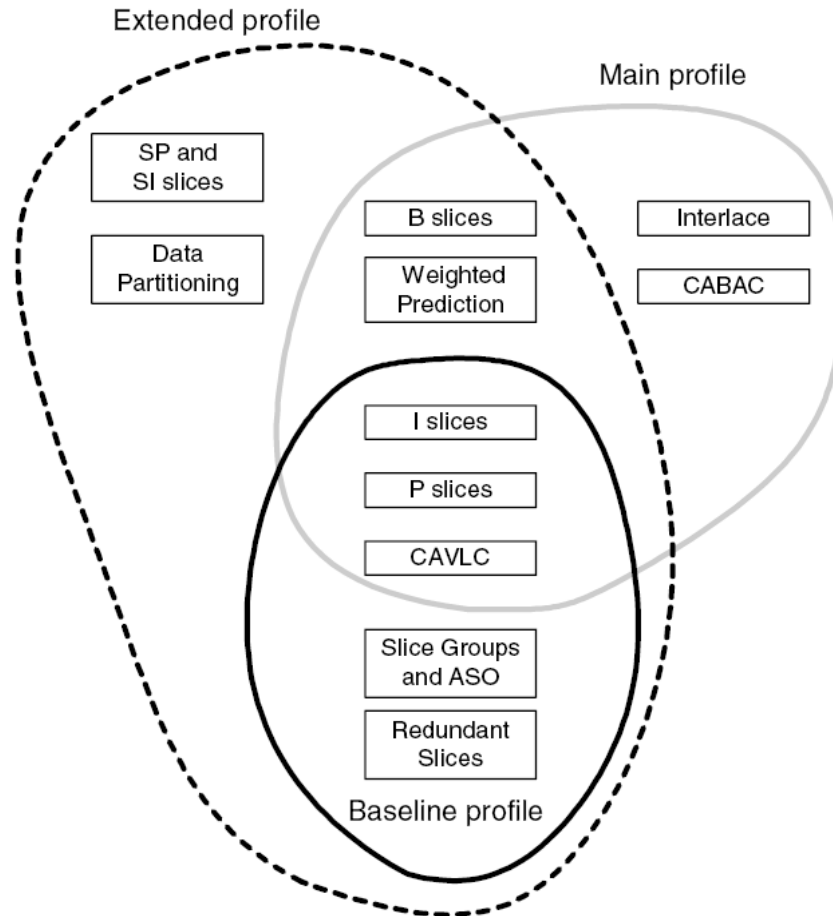
Application Requirements

- Coding efficiency
- Reliability
- Low-complexity
- Scalability
- Interlacing
- Low latency
- Lossless or near lossless
- Efficient transcoding

H.264 Levels & Profiles

- Levels
 - Restriction of resolution
 - Restriction of data throughput (Macroblocks/s)
- Profiles
 - Restriction of coding tools
 - Baseline, Main, High
 - Extended, FrExt → Error-resilience

H.264 Profiles



Dedicated HW extensions

- CAVLC / CABAC accelerator
- Pixel-based operations (SIMD, VLIW)
 - IDCT
 - Motion estimation / prediction
 - Filtering
- Intelligent DMA engines
 - 2D – transfers
 - Simple processing tasks (clipping, border expansion, etc.)

Standard Restrictions

- Application dependent
- Resolution dependent
 - 3GPP (Resolution, coding tools, etc.) → MMS
 - H.320 (ISDN-based video services)

IMPACT ON BITRATE!

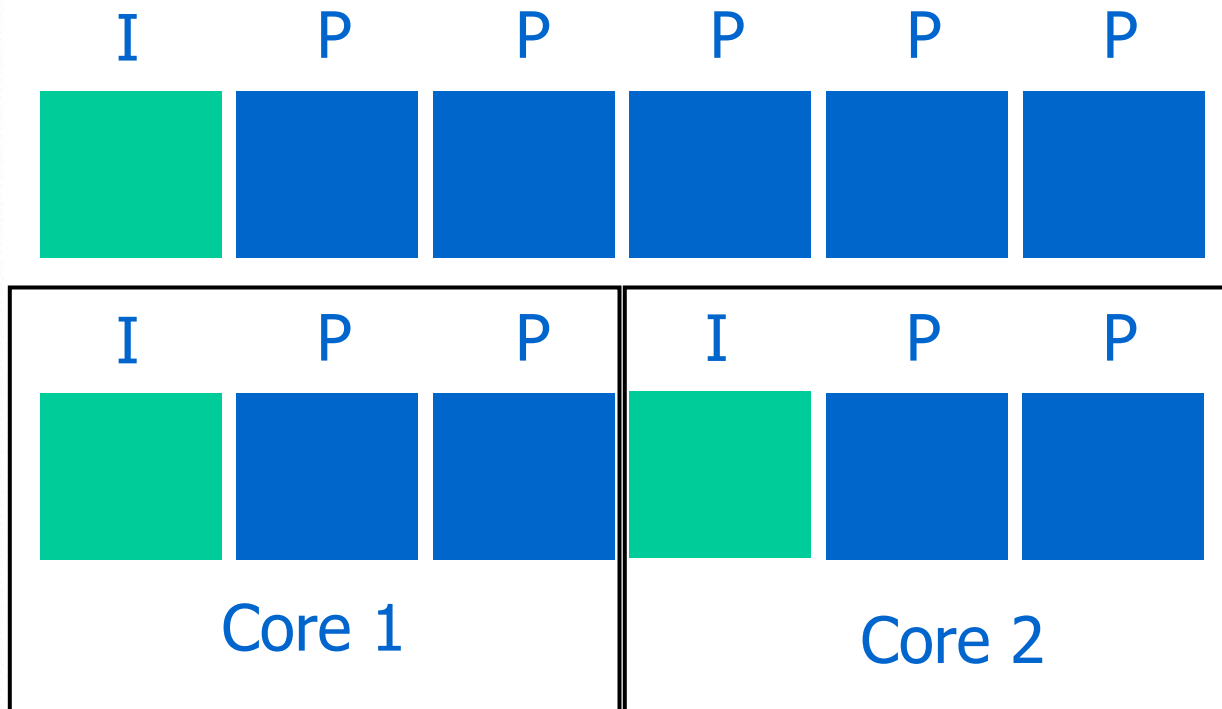
- Standard compliance required in H.264 decoder!

Reduction in Quality

- Simple ME algorithms
- Higher quantization
 - Reduces workload on entropy coder
- Only possible at encoder side!

IMPACT ON BITRATE!

Parallelization (Encoder)

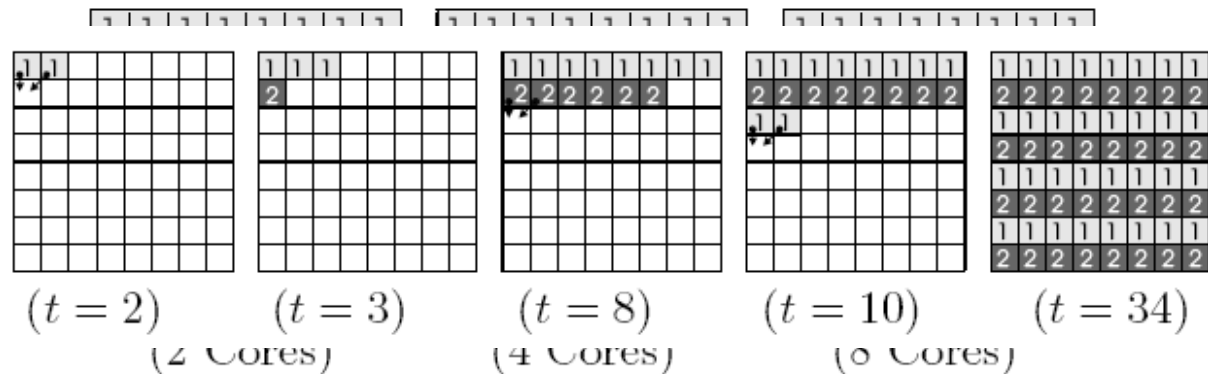


Parallel approaches (Decoder)

- Functional splitting
- Run decoding tasks on different CPUs
- Advantages
 - „Specialized“ blocks → typically smaller & less memory / cache
 - Strong algorithmic differences can be addressed by dedicted CPU extensions
 - Highly conditional parts (E.g. entropy coding, etc.)
 - Pixel-based signal processing parts (E.g. DCT, filtering)
- Problems
 - Unequal workload-balancing
 - Splitting interfaces must be defined & implemented
 - High data-transfers

Parallel decoder approaches (1)

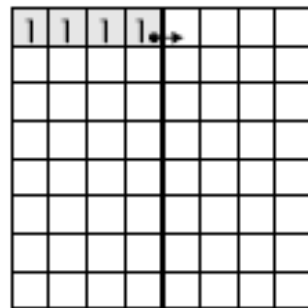
- Data-parallel approaches
 - Process luma / chroma data separately
 - Split frame into multiple regions and process them separately



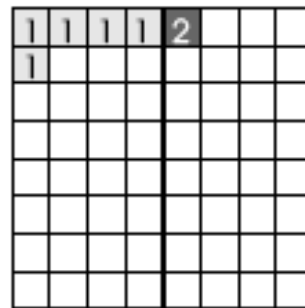
Encoder independent!

Parallel decoder approaches (2)

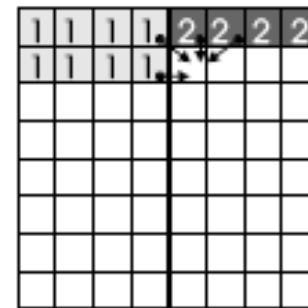
- Multi-column approach



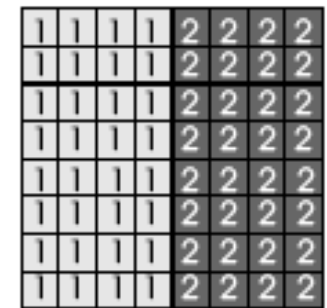
$(t = 4)$



$(t = 5)$



$(t = 8)$



$(t = 36)$

Encoder independent!

Thank you for your attention