IN5050:
Programming heterogeneous multi-core processors

Introduction to video coding



February 3, 2026

using slides by
Håvard Espeland, Pål Halvorsen,
Preben N. Olsen, Carsten Griwodz

Why ?

IN5050 is about programming heterogeneous multi-core processors.

Why an intro to video coding?

Parallel processing without workloads has always targeted individual algorithms: counting, searching, sorting, ...

Real workloads use multiple, different algorithms that need to cooperate and be interconnected and have non-parallelizable steps as well

We want to look at parallelism that is required for everyday tasks, not at super-high-end scientific computing or incomprehensible machine learning stages

We know a bit more about video streaming than ot

... and Håkon doesn't want me to use SIFT for IN5050 instead

- Many of its applications are time-critical
- A codec can become memory-bound, CPU-bound, IO-bound
- A single codec has opportunities for data parallelism, functional parallelism, and pipelining
- It is also a very important workload in its own right

Why ?

IN5050 is about programming heterogeneous multi-core processors

Why an intro to video coding?

Parallel processing without ... ithms: counting, searching, s...

Real workloads ... d be interconnected ...

We want to look ... r-high-end scientific com...

We know a bit more ... sks:

- Many of its applica...
- A codec can becom... bound, IO-bound
- A single codec has op... data parallelism, functional parallelism, and pipelining
- It is also a very important workload in its own right

According to Cisco predictions in 2019, Internet video globally
- ... consumed 159 Exabytes per month in 2019.
- ... had a compound annual growth rate of 31%.
- ... was expected to reach 280 exabytes/month in 2022.
- ... had 82% of consumer Internet traffic in 2021.
- ...

Statista reports number of users
- >3900 mio over-the-top (OTT) streaming users in 2024[1]
- Expect > 5110 mio users in 2030[2]

([1] 2024 information, [2] 2026 prediction)

Data Compression

- Alternative description of data requiring less storage and bandwidth



Uncompressed: **1 Mbyte**



Compressed (JPEG)**: 50 Kbyte** (20:1)

... while, for example, a 20 Megapixel camera creates 6016 x 4000 images,
in 8-bit RGB that makes more than 72 uncompressed Mbytes per image

IN5050:
Programming heterogeneous multi-core processors

Coding pictures



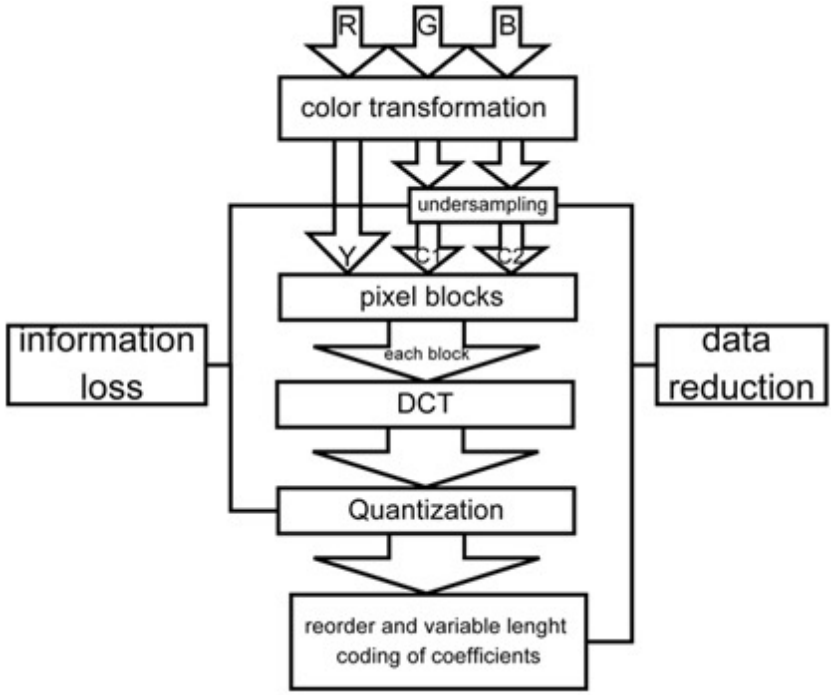
„Kodak No 2-A Folding Autographic Brownie (ca 1917)" by Carsten Corleis – own work. Licensed under Creative Commons Attribution-Share Alike 3.0

JPEG

- "JPEG": Joint Photographic Expert Group

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

- Compression rate of 1:10 yields reasonable results
  (better rates are also possible)



Color conversion: RGB to YCbCr

$$Y = 0.299 \cdot R + 0.587 \cdot G + 0.114B$$
$$Cb = 128 - 0.168736R - 0.331264G + 0.5B$$
$$Cr = 128 + 0.5R - 0.418688G - 0.081312B$$

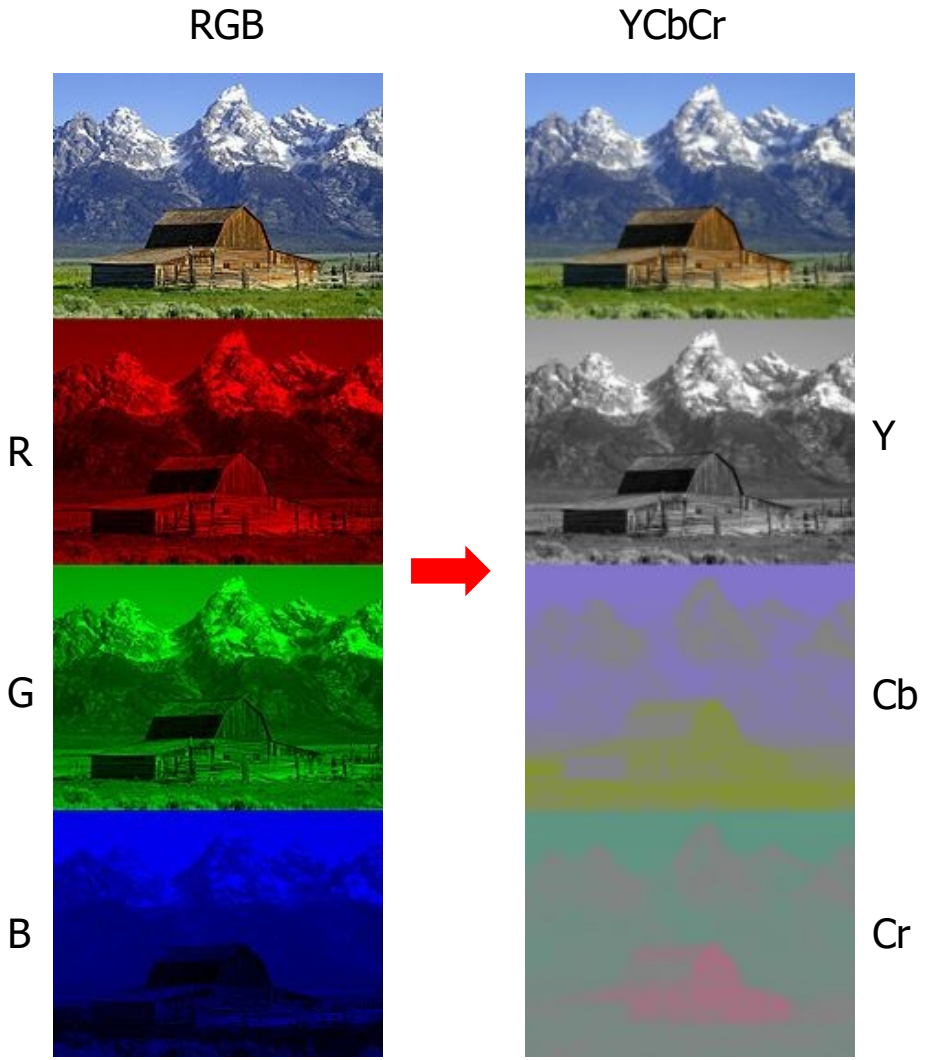
Y image is a greyscale version of the main image

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

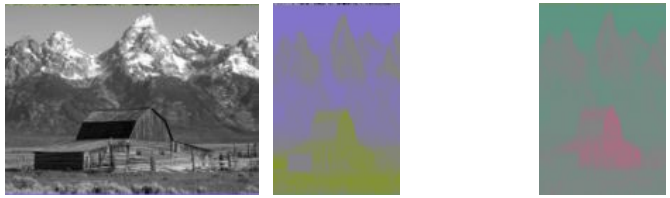
the blue sky is represented by strong Cb and weak Cr



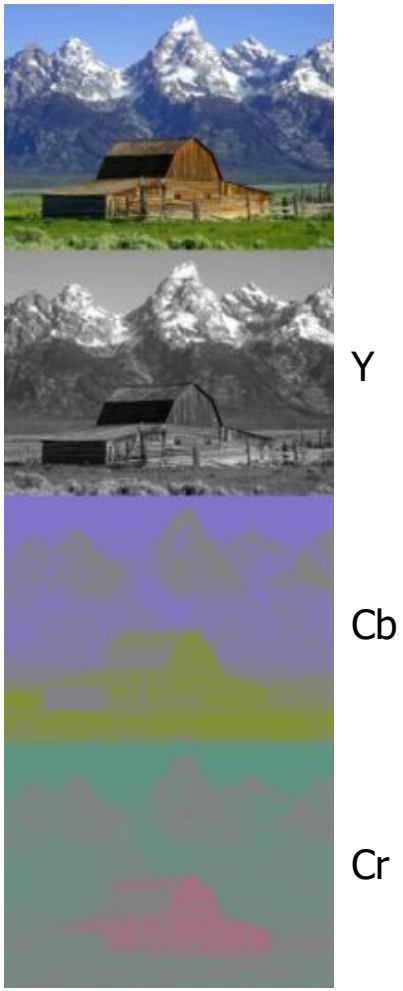
(Chroma) Downsampling

- Downsampling (first compression step)
  - humans can see considerably more fine detail in the brightness, i.e., can reduce the spatial resolution of the Cb and Cr components
  - 4:4:4 (no downsampling)
  - 4:2:2 (reduce by factor of 2 in horizontal direction)
  - 4:2:0 (reduce by factor of 2 in horizontal and vertical directions)



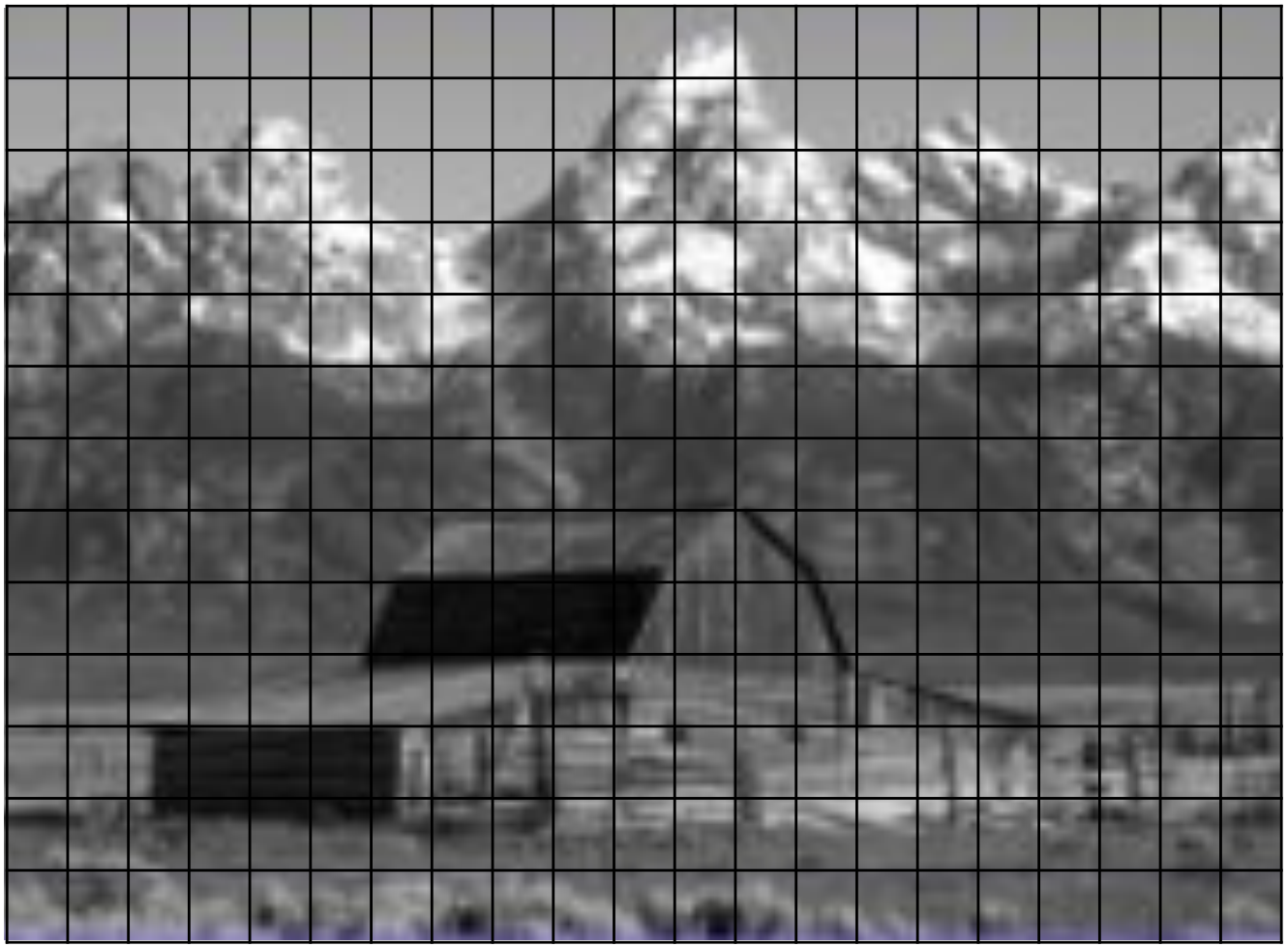






Split each picture in 8x8 blocks

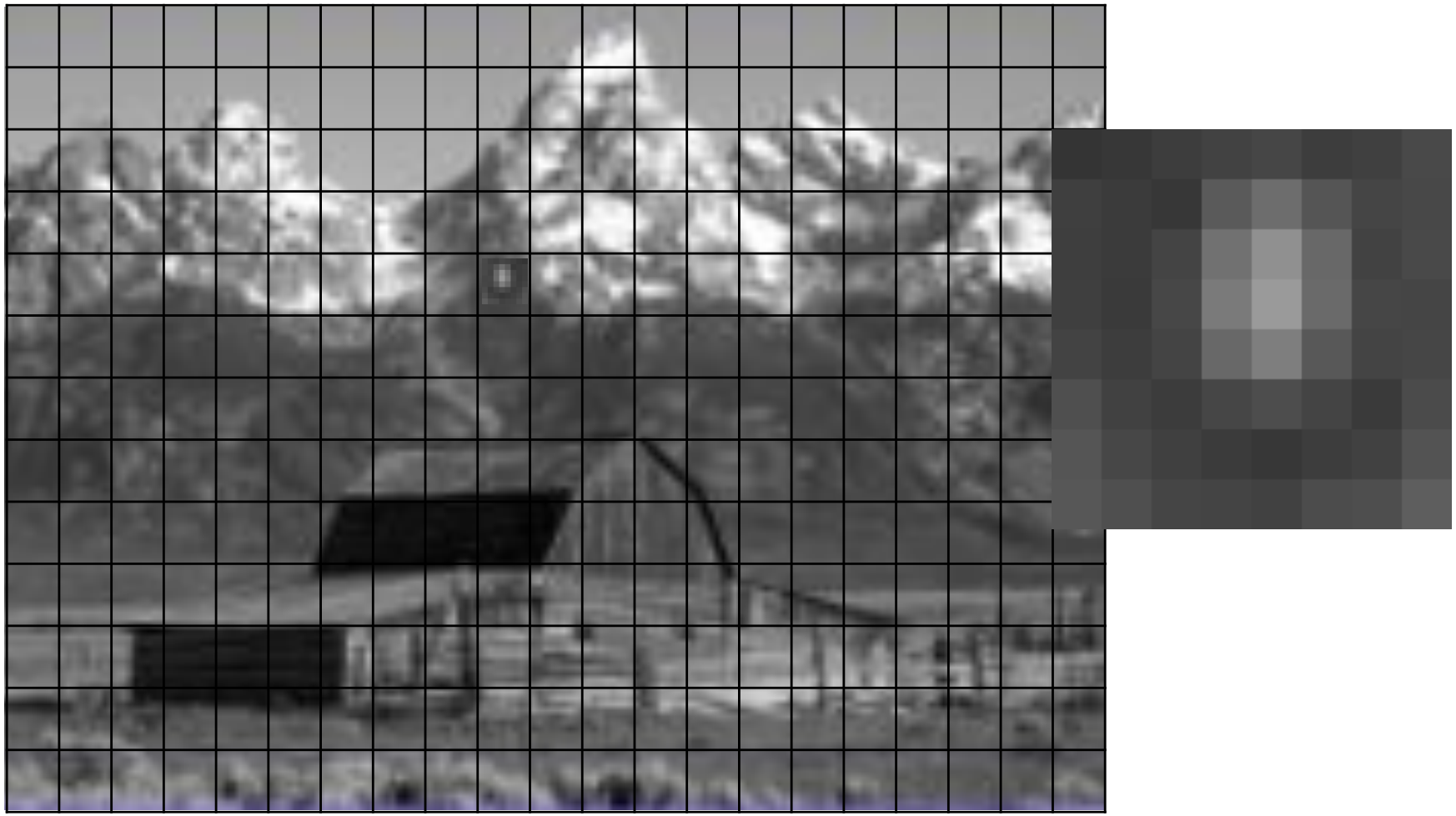
- **Each** Y, Cb and Cr picture is divided into 8x8 blocks



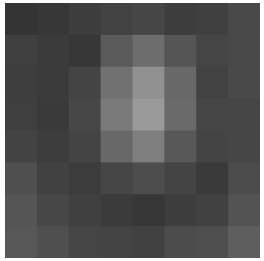


Discrete cosine transform (DCT)

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



DCT



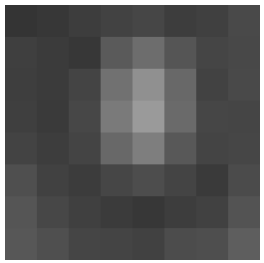
$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} - 128 = g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

($x \rightarrow$, $y \downarrow$)

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

- initially, each pixel is represented by a [0, 255]-value
- each pixel is transformed to a [-128, 127]-value

DCT



$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} - 128 = g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

$x \longrightarrow$ $y \downarrow$

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

- two-dimensional DCT: $$G_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} g_{x,y}\cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$
  - $G_{u,v}$ is the DCT at output coordinates (u,v) in the transformed 8x8 matrix
  - u and v are from $\{0, \ldots, 7\}$
  - $g_{x,y}$ is the pixel value at input coordinates (x,y)
  - α is a normalizing function: $$\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

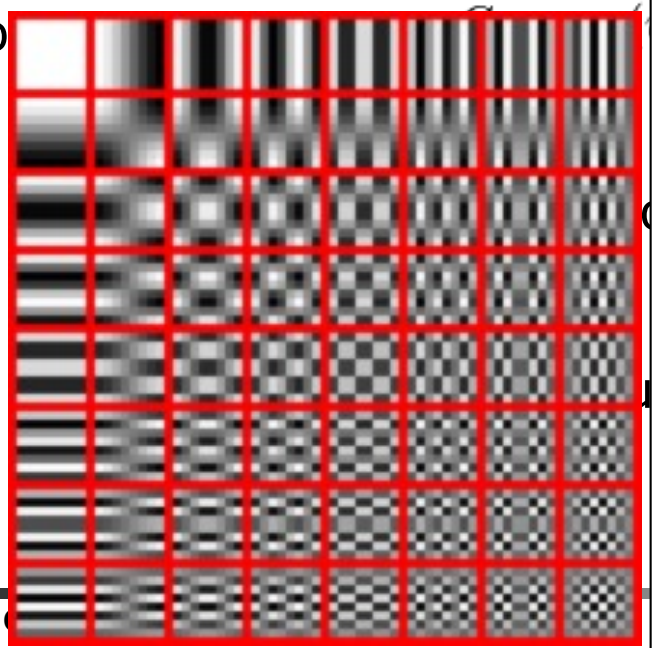
DCT

$$g_{x,y} =$$

	0	1	2	3	4	5	6	7
0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
1	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
2	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
3	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
4	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
5	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
6	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
7	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0

Each 8 × 8 block (Y, Cb, Cr) is conve
using a normalized, two-dimensiona

- two



u \ v	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1
4	1	1	1	1	1	1	1	1
5	1	1	1	1	1	1	1	1
6	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1

u \ v	0	1	2	3	4	5	6	7
0	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
1	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
2	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
3	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
4	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
5	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
6	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0
7	1,0	0,8	0,6	0,2	-0,2	-0,6	-0,8	-1,0

u \ v	0	1	2	3	4	5	6	7
0	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
1	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
2	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
3	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
4	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
5	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
6	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
7	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9

u \ v	0	1	2	3	4	5	6	7
0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
1	0,8	0,8	0,8	0,8	0,8	0,8	0,8	0,8
2	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6
3	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2
4	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
5	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6
6	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8
7	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0

u \ v	0	1	2	3	4	5	6	7
0	1,0	0,8	0,5	0,2	-0,2	-0,5	-0,8	-1,0
1	0,8	0,7	0,5	0,2	-0,2	-0,5	-0,7	-0,8
2	0,5	0,5	0,3	0,1	-0,1	-0,3	-0,5	-0,5
3	0,2	0,2	0,1	0,0	0,0	-0,1	-0,2	-0,2
4	-0,2	-0,2	-0,1	0,0	0,0	0,1	0,2	0,2
5	-0,5	-0,5	-0,3	-0,1	0,1	0,3	0,5	0,5
6	-0,8	-0,7	-0,5	-0,2	0,2	0,5	0,7	0,8
7	-1,0	-0,8	-0,5	-0,2	0,2	0,5	0,8	1,0

u \ v	0	1	2	3	4	5	6	7
0	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
1	0,8	0,3	-0,3	-0,8	-0,8	-0,3	0,3	0,8
2	0,5	0,2	-0,2	-0,5	-0,5	-0,2	0,2	0,5
3	0,2	0,1	-0,1	-0,2	-0,2	-0,1	0,1	0,2
4	-0,2	-0,1	0,1	0,2	0,2	0,1	-0,1	-0,2
5	-0,5	-0,2	0,2	0,5	0,5	0,2	-0,2	-0,5
6	-0,8	-0,3	0,3	0,8	0,8	0,3	-0,3	-0,8
7	-0,9	-0,4	0,4	0,9	0,9	0,4	-0,4	-0,9

u \ v	0	1	2	3	4	5	6	7
0	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9
1	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4
2	-0,4	-0,4	-0,4	-0,4	-0,4	-0,4	-0,4	-0,4
3	-0,9	-0,9	-0,9	-0,9	-0,9	-0,9	-0,9	-0,9
4	-0,9	-0,9	-0,9	-0,9	-0,9	-0,9	-0,9	-0,9
5	-0,4	-0,4	-0,4	-0,4	-0,4	-0,4	-0,4	-0,4
6	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4
7	0,9	0,9	0,9	0,9	0,9	0,9	0,9	0,9

u \ v	0	1	2	3	4	5	6	7
0	0,9	0,8	0,5	0,2	-0,2	-0,5	-0,8	-0,9
1	0,4	0,3	0,2	0,1	-0,1	-0,2	-0,3	-0,4
2	-0,4	-0,3	-0,2	-0,1	0,1	0,2	0,3	0,4
3	-0,9	-0,8	-0,5	-0,2	0,2	0,5	0,8	0,9
4	-0,9	-0,8	-0,5	-0,2	0,2	0,5	0,8	0,9
5	-0,4	-0,3	-0,2	-0,1	0,1	0,2	0,3	0,4
6	0,4	0,3	0,2	0,1	-0,1	-0,2	-0,3	-0,4
7	0,9	0,8	0,5	0,2	-0,2	-0,5	-0,8	-0,9

u \ v	0	1	2	3	4	5	6	7
0	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
1	0,4	0,1	-0,1	-0,4	-0,4	-0,1	0,1	0,4
2	-0,4	-0,1	0,1	0,4	0,4	0,1	-0,1	-0,4
3	-0,9	-0,4	0,4	0,9	0,9	0,4	-0,4	-0,9
4	-0,9	-0,4	0,4	0,9	0,9	0,4	-0,4	-0,9
5	-0,4	-0,1	0,1	0,4	0,4	0,1	-0,1	-0,4
6	0,4	0,1	-0,1	-0,4	-0,4	-0,1	0,1	0,4
7	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9

u \ v	0	1	2	3	4	5	6	7
0	0,8	0,8	0,8	0,8	0,8	0,8	0,8	0,8
1	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
2	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0
3	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6
4	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6
5	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
6	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2
7	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8

u \ v	0	1	2	3	4	5	6	7
0	0,8	0,7	0,5	0,2	-0,2	-0,5	-0,7	-0,8
1	-0,2	-0,2	-0,1	0,0	0,0	0,1	0,2	0,2
2	-1,0	-0,8	-0,5	-0,2	0,2	0,5	0,8	1,0
3	-0,5	-0,5	-0,3	-0,1	0,1	0,3	0,5	0,5
4	0,5	0,5	0,3	0,1	-0,1	-0,3	-0,5	-0,5
5	1,0	0,8	0,5	0,2	-0,2	-0,5	-0,8	-1,0
6	0,2	0,2	0,1	0,0	0,0	-0,1	-0,2	-0,2
7	-0,8	-0,7	-0,5	-0,2	0,2	0,5	0,7	0,8

u \ v	0	1	2	3	4	5	6	7
0	0,8	0,3	-0,3	-0,8	-0,8	-0,3	0,3	0,8
1	-0,2	-0,1	0,1	0,2	0,2	0,1	-0,1	-0,2
2	-0,9	-0,4	0,4	0,9	0,9	0,4	-0,4	-0,9
3	-0,5	-0,2	0,2	0,5	0,5	0,2	-0,2	-0,5
4	0,5	0,2	-0,2	-0,5	-0,5	-0,2	0,2	0,5
5	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
6	0,2	0,1	-0,1	-0,2	-0,2	-0,1	0,1	0,2
7	-0,8	-0,3	0,3	0,8	0,8	0,3	-0,3	-0,8

u \ v	0	1	2	3	4	5	6	7
0	0,7	0,7	0,7	0,7	0,7	0,7	0,7	0,7
1	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7
2	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7
3	0,7	0,7	0,7	0,7	0,7	0,7	0,7	0,7
4	0,7	0,7	0,7	0,7	0,7	0,7	0,7	0,7
5	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7
6	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7	-0,7
7	0,7	0,7	0,7	0,7	0,7	0,7	0,7	0,7

u \ v	0	1	2	3	4	5	6	7
0	0,7	0,6	0,4	0,1	-0,1	-0,4	-0,6	-0,7
1	-0,7	-0,6	-0,4	-0,1	0,1	0,4	0,6	0,7
2	-0,7	-0,6	-0,4	-0,1	0,1	0,4	0,6	0,7
3	0,7	0,6	0,4	0,1	-0,1	-0,4	-0,6	-0,7
4	0,7	0,6	0,4	0,1	-0,1	-0,4	-0,6	-0,7
5	-0,7	-0,6	-0,4	-0,1	0,1	0,4	0,6	0,7
6	-0,7	-0,6	-0,4	-0,1	0,1	0,4	0,6	0,7
7	0,7	0,6	0,4	0,1	-0,1	-0,4	-0,6	-0,7

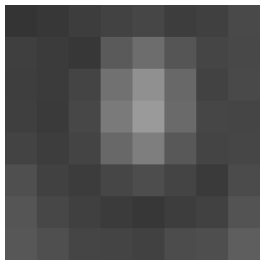
u \ v	0	1	2	3	4	5	6	7
0	0,7	0,3	-0,3	-0,7	-0,7	-0,3	0,3	0,7
1	-0,7	-0,3	0,3	0,7	0,7	0,3	-0,3	-0,7
2	-0,7	-0,3	0,3	0,7	0,7	0,3	-0,3	-0,7
3	0,7	0,3	-0,3	-0,7	-0,7	-0,3	0,3	0,7
4	0,7	0,3	-0,3	-0,7	-0,7	-0,3	0,3	0,7
5	-0,7	-0,3	0,3	0,7	0,7	0,3	-0,3	-0,7
6	-0,7	-0,3	0,3	0,7	0,7	0,3	-0,3	-0,7
7	0,7	0,3	-0,3	-0,7	-0,7	-0,3	0,3	0,7

u \ v	0	1	2	3	4	5	6	7
0	0,6	0,6	0,6	0,6	0,6	0,6	0,6	0,6
1	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0	-1,0
2	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2
3	0,8	0,8	0,8	0,8	0,8	0,8	0,8	0,8
4	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8	-0,8
5	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2	-0,2
6	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
7	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6	-0,6

u \ v	0	1	2	3	4	5	6	7
0	0,5	0,5	0,3	0,1	-0,1	-0,3	-0,5	-0,5
1	-1,0	-0,8	-0,5	-0,2	0,2	0,5	0,8	1,0
2	0,2	0,2	0,1	0,0	0,0	-0,1	-0,2	-0,2
3	0,8	0,7	0,5	0,2	-0,2	-0,5	-0,7	-0,8
4	-0,8	-0,7	-0,5	-0,2	0,2	0,5	0,7	0,8
5	-0,2	-0,2	-0,1	0,0	0,0	0,1	0,2	0,2
6	1,0	0,8	0,5	0,2	-0,2	-0,5	-0,8	-1,0
7	-0,5	-0,5	-0,3	-0,1	0,1	0,3	0,5	0,5

u \ v	0	1	2	3	4	5	6	7
0	0,5	0,2	-0,2	-0,5	-0,5	-0,2	0,2	0,5
1	-0,9	-0,4	0,4	0,9	0,9	0,4	-0,4	-0,9
2	0,2	0,1	-0,1	-0,2	-0,2	-0,1	0,1	0,2
3	0,8	0,3	-0,3	-0,8	-0,8	-0,3	0,3	0,8
4	-0,8	-0,3	0,3	0,8	0,8	0,3	-0,3	-0,8
5	-0,2	-0,1	0,1	0,2	0,2	0,1	-0,1	-0,2
6	0,9	0,4	-0,4	-0,9	-0,9	-0,4	0,4	0,9
7	-0,5	-0,2	0,2	0,5	0,5	0,2	-0,2	-0,5

DCT



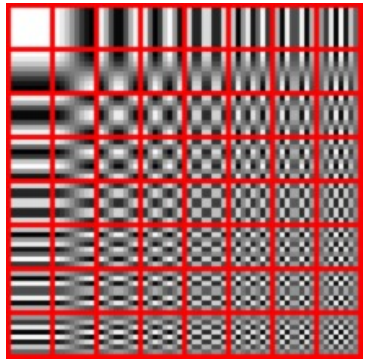
$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix} - 128 = g = \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \begin{matrix} x \longrightarrow \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \quad \downarrow y$$

Apply DCT =

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

Note the rather large value of the top-left corner (DC coefficient). The remaining 63 are AC coefficients. The advantage of the DCT is its tendency to aggregate most of the signal (the block average) in one corner of the result, as may be seen to the right.

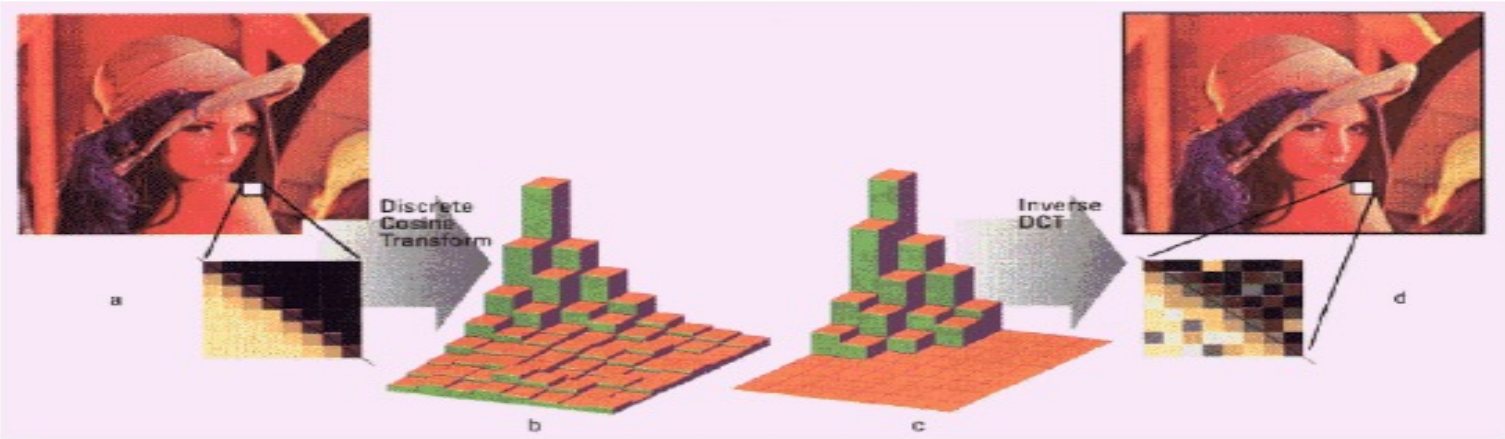
Compression possible: the following **quantization** step accentuates this effect while simultaneously reducing the overall size of the DCT coefficients



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 $Q_{j,k}$ is a quantization matrix

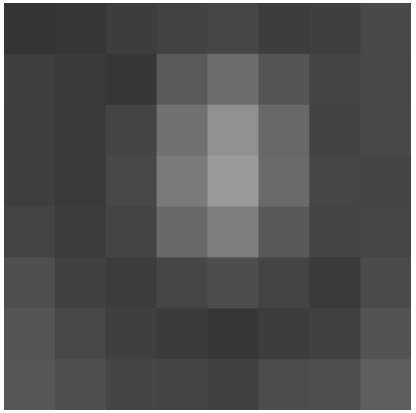
Quantization Example

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

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

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

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

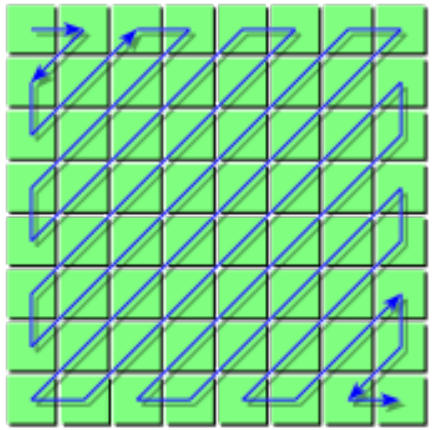


Lossless compression

The resulting data for all 8 × 8 blocks is further compressed with a loss-less algorithm

1. organize numbers in **zigzag pattern**

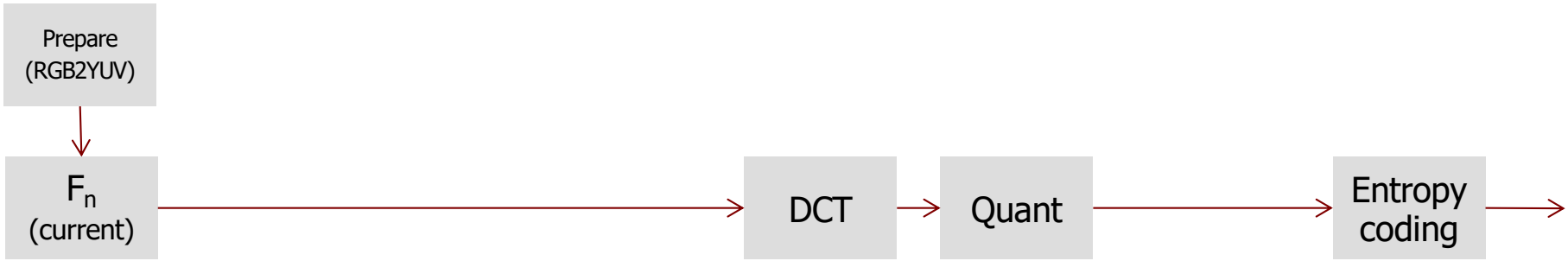
$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



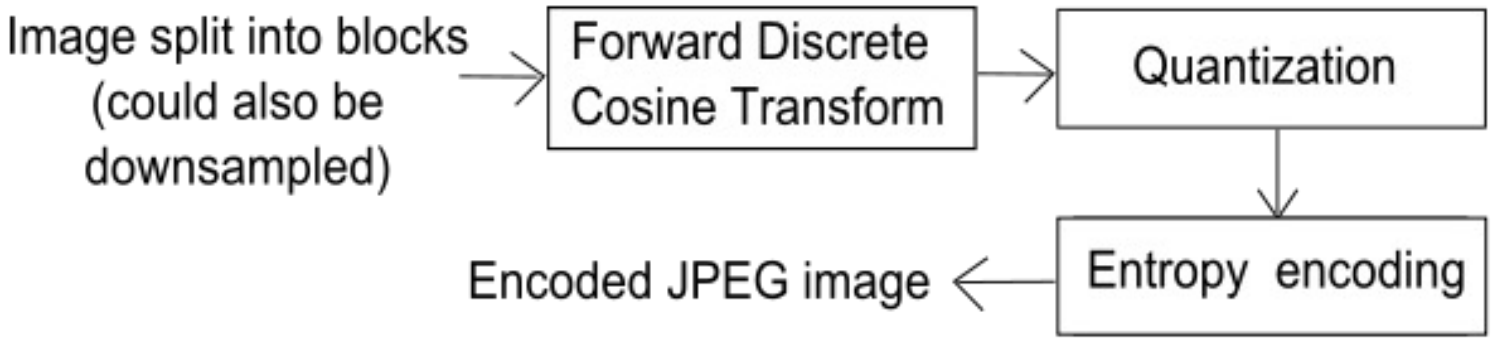
➔ -26, -3, 0, -3, -2, -6, 2, -4, 1, -4, 1, 1, 5, 1, 2, -1, 1, -1, 2, 0, 0, 0, 0, 0, -1, -1, 0 , 0, 0, 0, 0, 0, 0, 0, 0, …., 0, 0

2. run-length coding

JPEG Encoder Overview



Encoding:



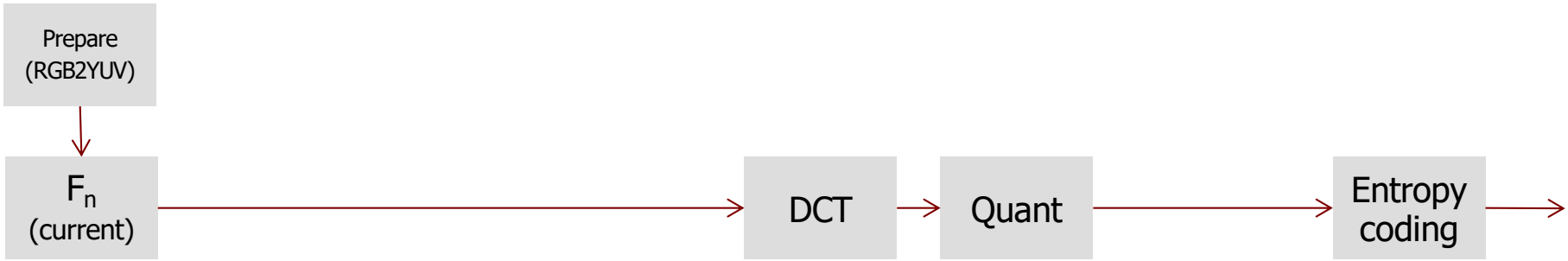
IN5050:
Programming heterogeneous multi-core processors

Coding videos



"Pathé-Baby hand movie camera (right)" by Cquoi - Own work.
Licensed under Creative Commons Attribution-Share Alike 3.0

Motion JPEG



- Motion JPEG
  - compress video by repeating JPEG steps
  - allows pipelining of frames
- Motion JPEG is not a standard
  - Apple has a Quicktime container
  - Microsoft has another container
  - the IETF has defined an RTP profile
  - ...
  - and they are all (mostly) incompatible

Still very typical for video editing tools

(but nothing else)

A picture



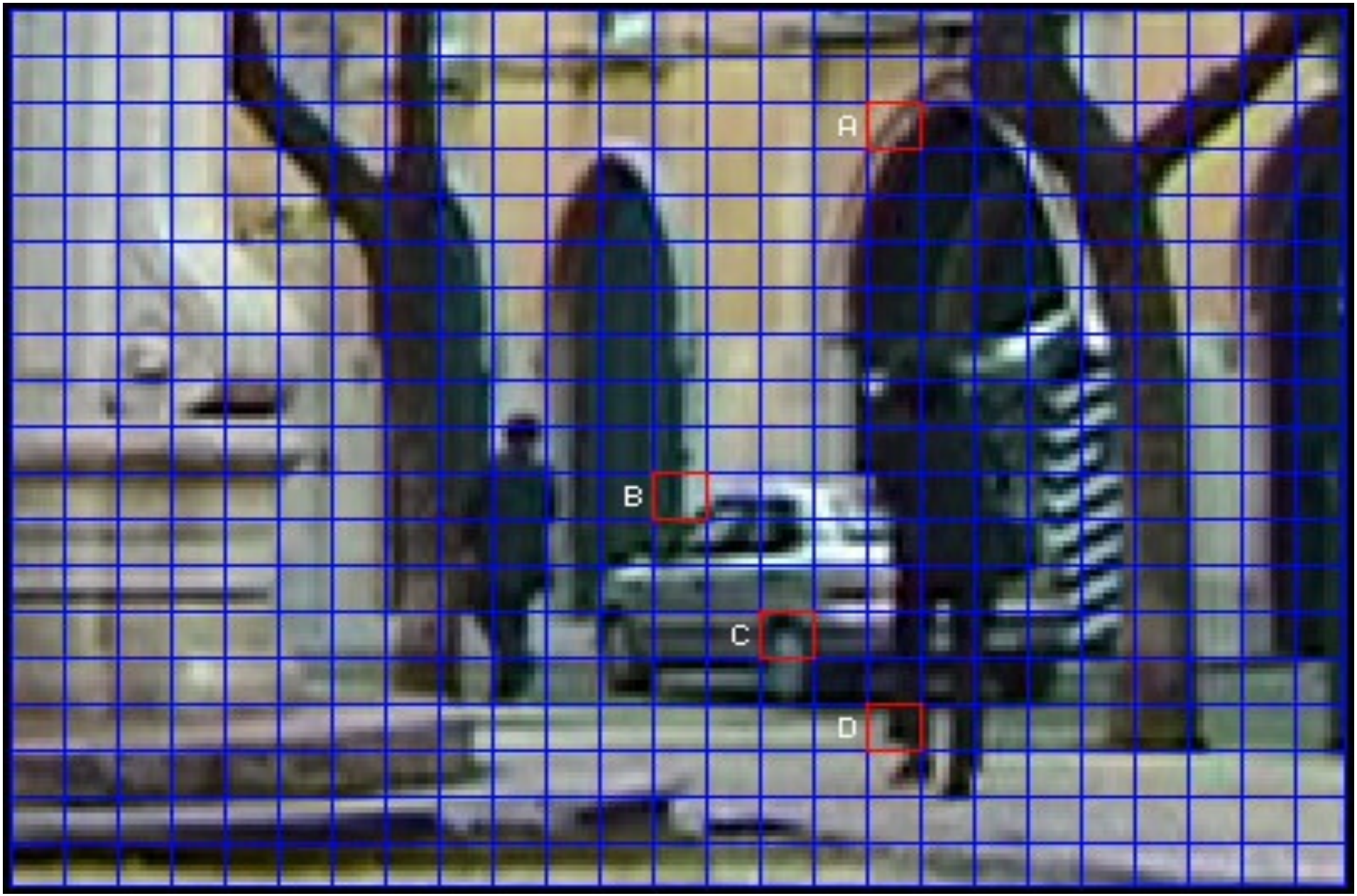
A new pictures? A changed picture!



Macro blocks



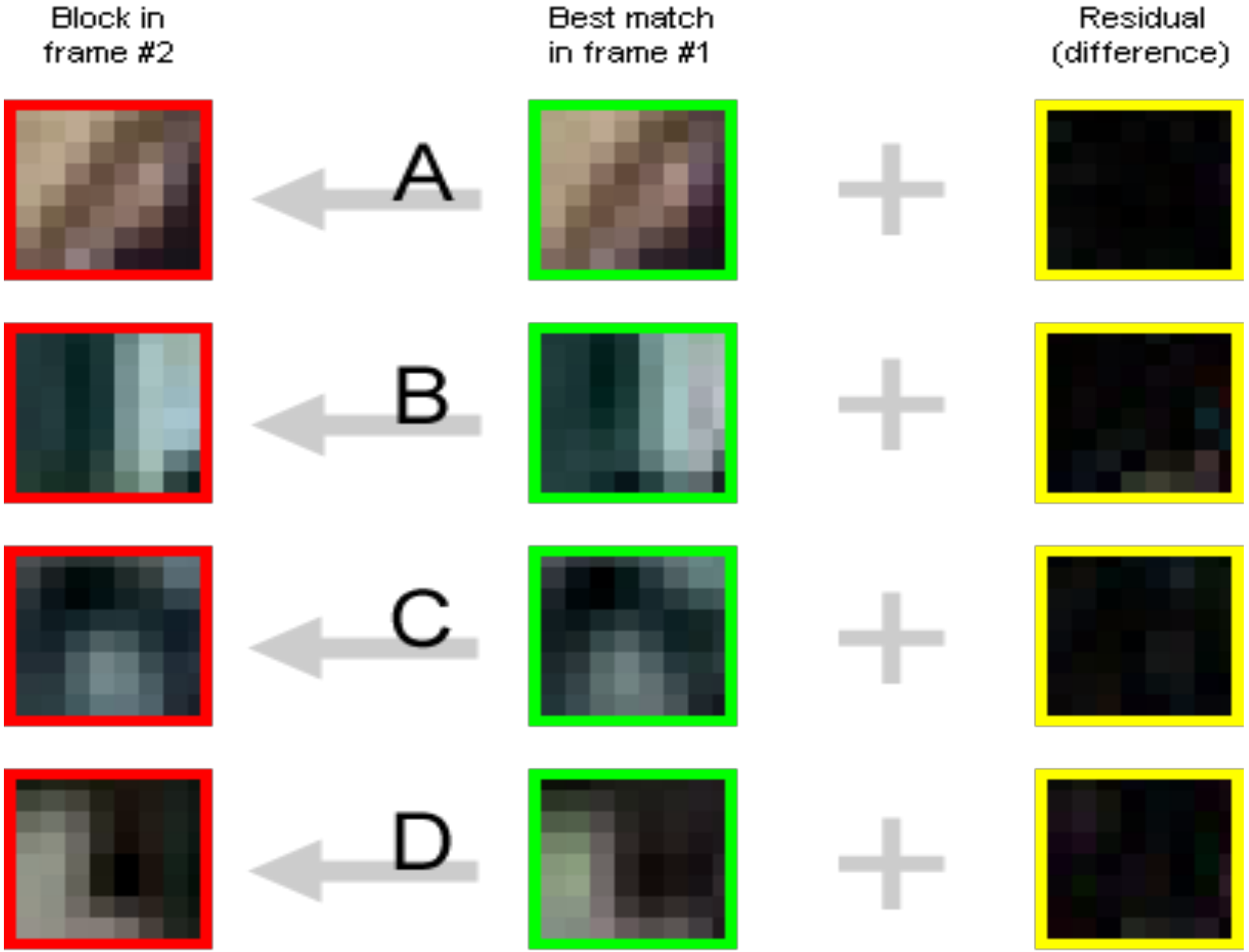
Focusing on blocks A B C & D



Best match in reference frame



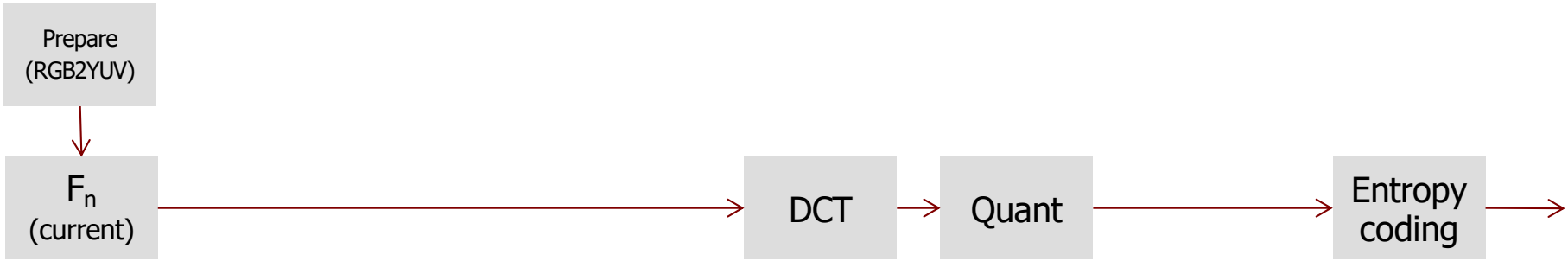
1. Challenge: small differences



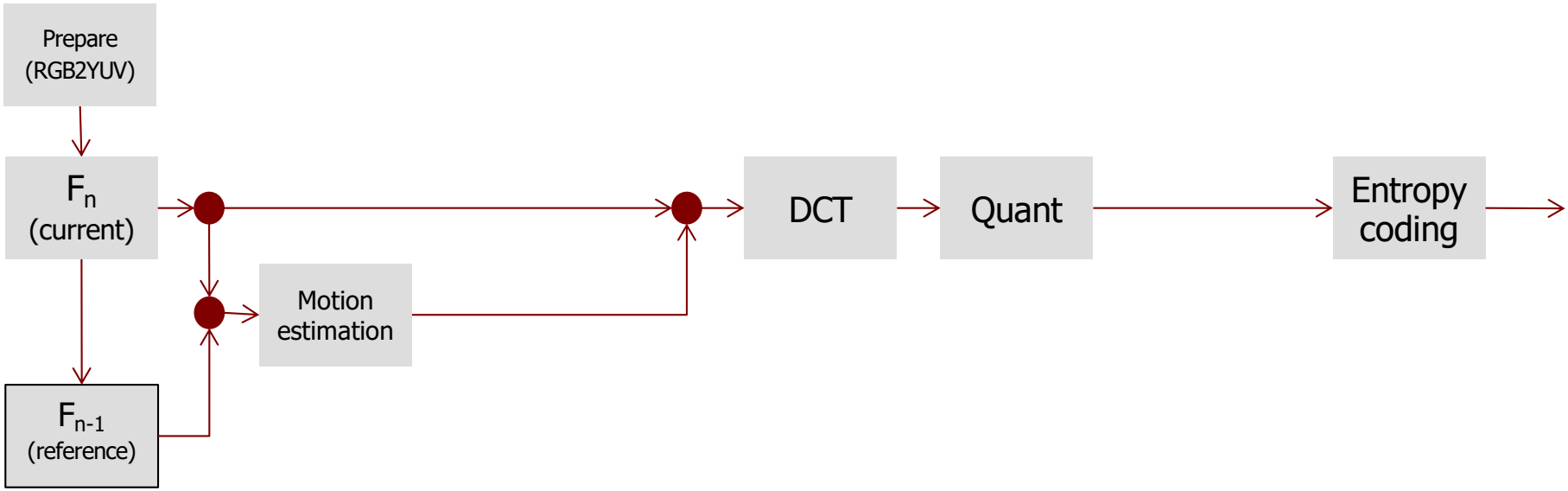
Challenge: Displacement



Adding Motion to the JPEG pipeline

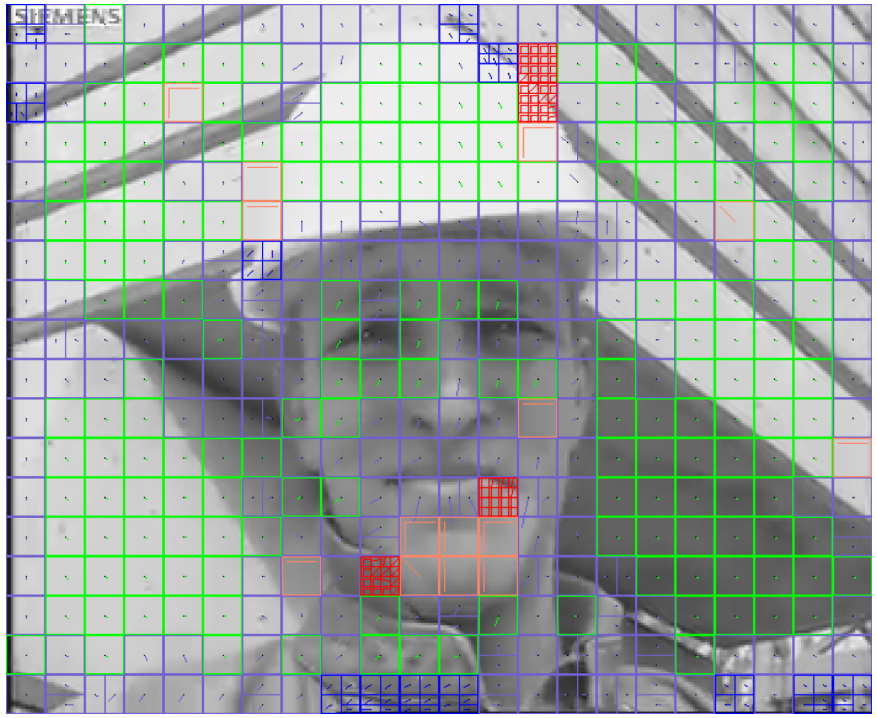


Adding Motion to the JPEG pipeline

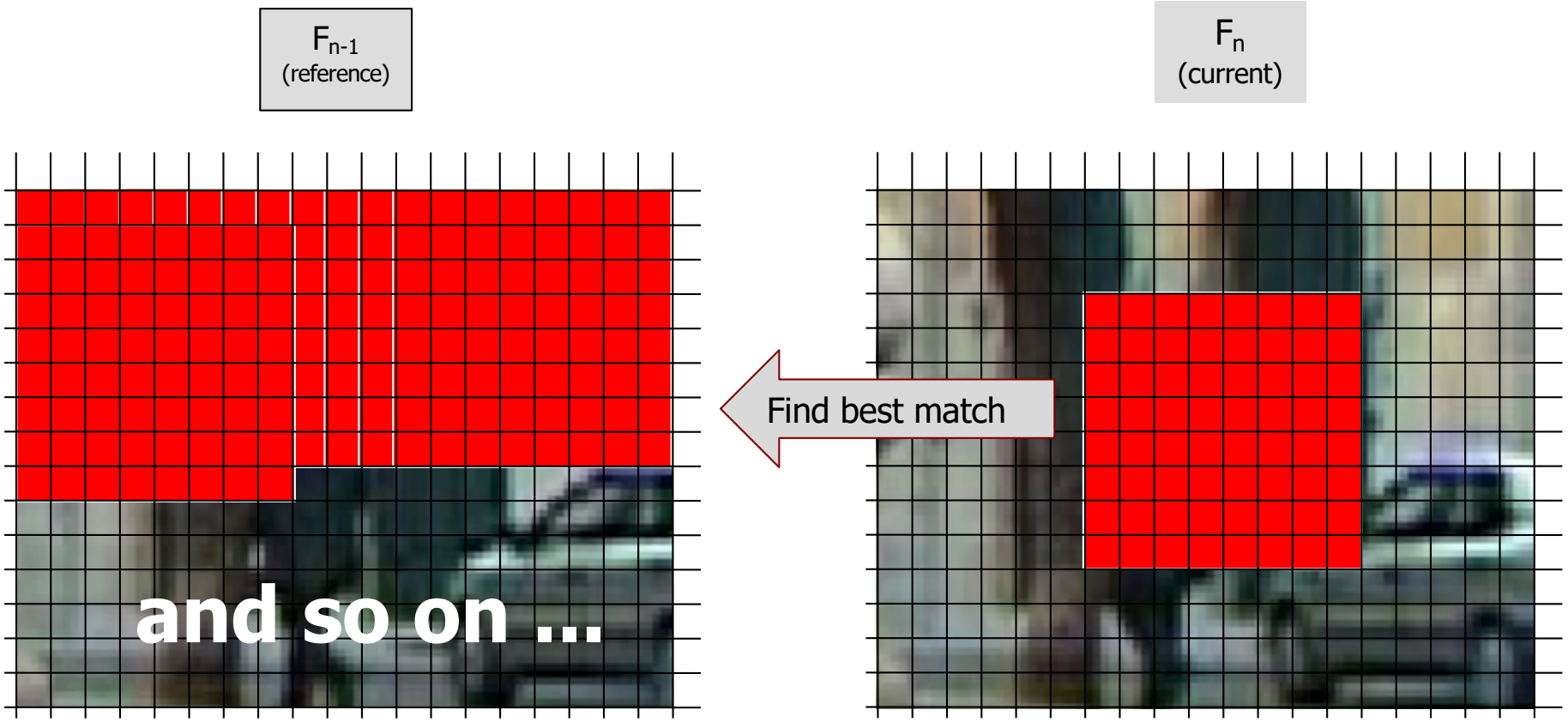


Inter-Prediction

- Predict a macroblock by reusing pixels from another frame
  - *Motion vectors* are used to compensate for movement
  - H.263: search in 1 frame
  - MPEG-1/2: search in 2 frames
  - H.264 (AVC) + H.265 (HEVC): search in 16 frames



Full Search Motion Estimation



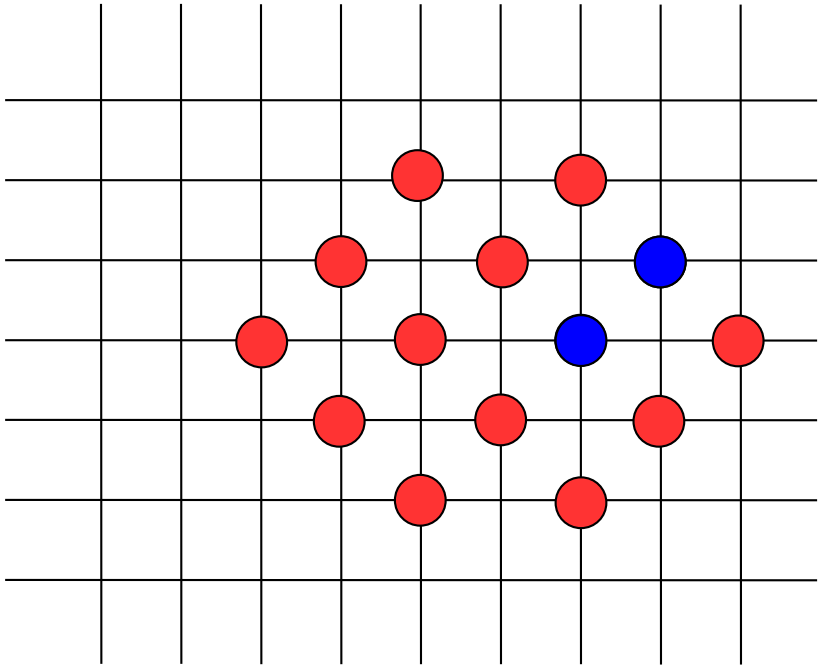
For comparing blocks:

- SAD - Sum of Absolute Differences

$$\sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i, y+j)|$$

W: fixed set, but not only integers !

Diamond Motion Estimation Pattern



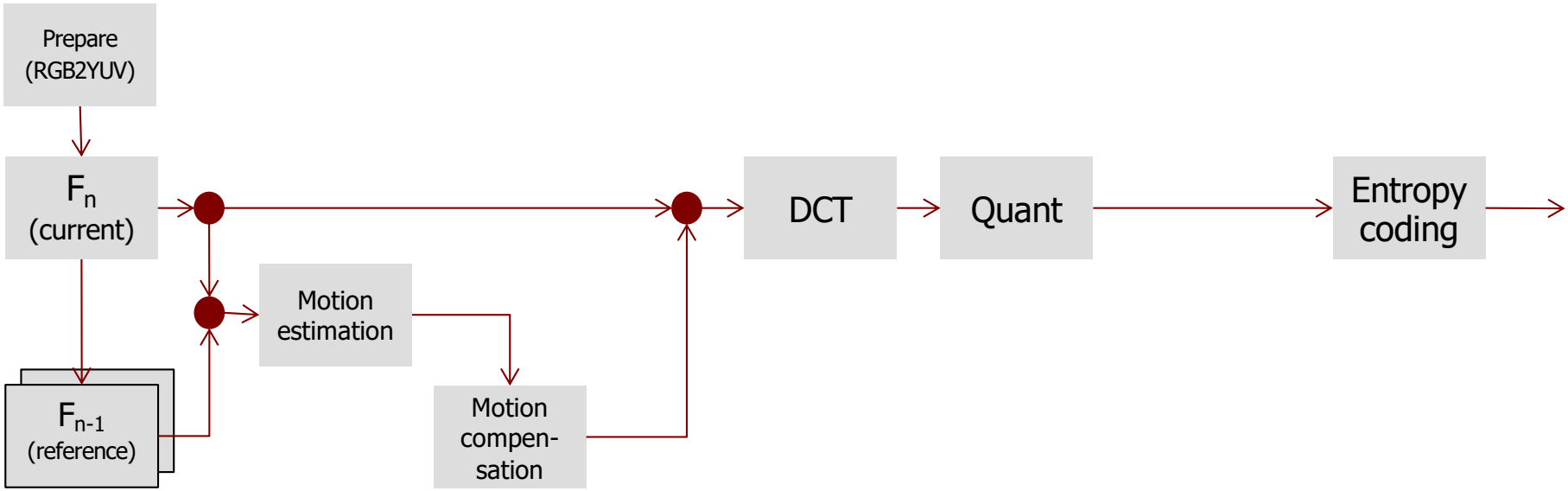
Heuristic

- start at the same place in previous frame
- search neighourhood first
- move search center to location with smallest SAD
- repeat until stable

Motion Estimation

- The estimators often use a two-step process, with initial coarse evaluation and refinements
- Don’t do this for every frame, you must sometimes encode macroblocks in a “safe” mode that doesn’t rely on others
- This is called “Intra”-mode
  - When a complete frame is encoded in I-mode (always in MPEG-1 and MPEG-2), this is called an I-frame
  - x264 calls I-frames "keyframes". But the word keyframe has many, many other meanings as well. Avoid misunderstandings. Write I-frame. There are also IBR frames.
- Refinements include trying every block in the area, and also using sub-pixel precision (interpolation)
  - quarter pixel in H.264

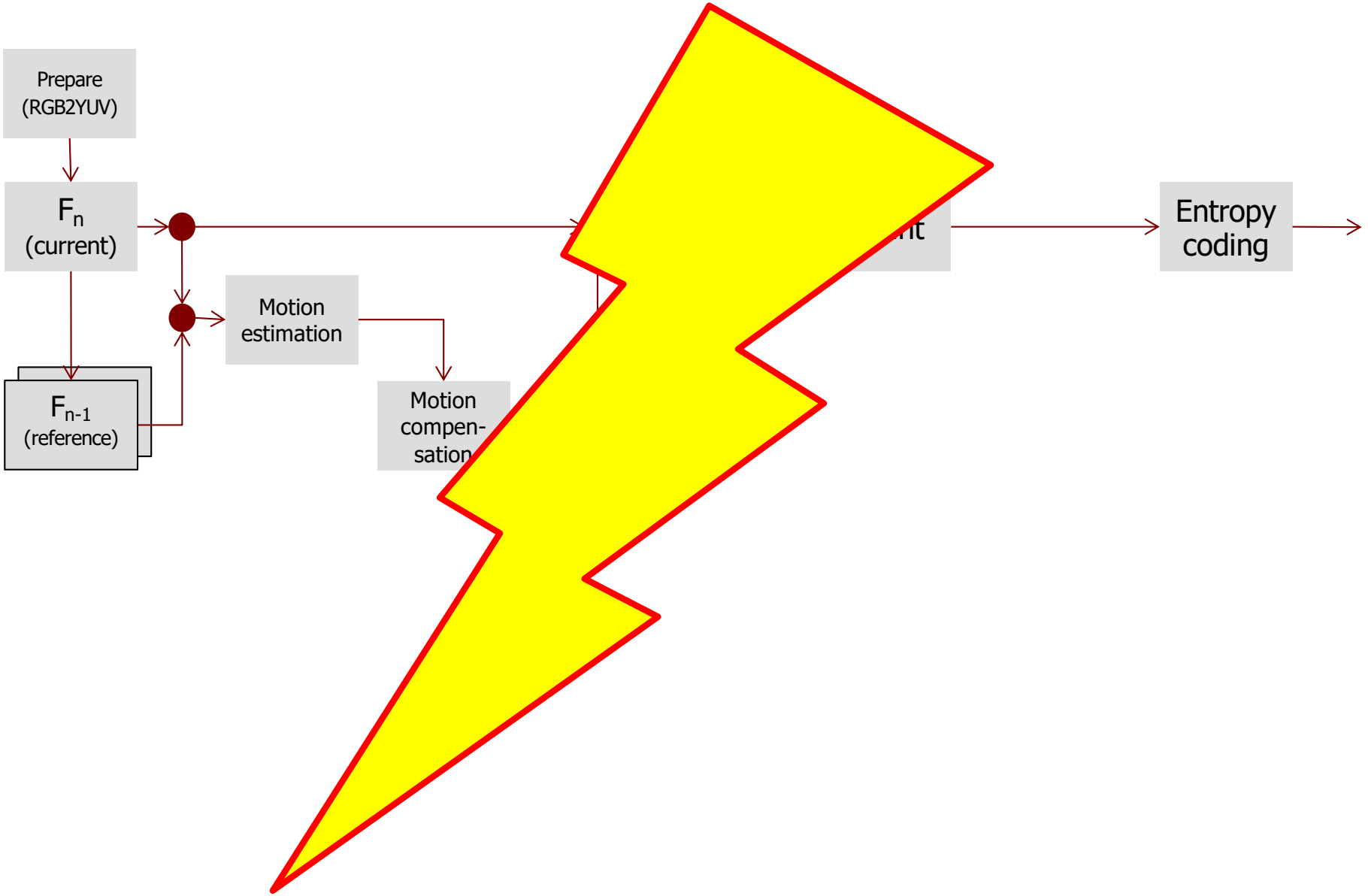
Adding Motion to the JPEG pipeline



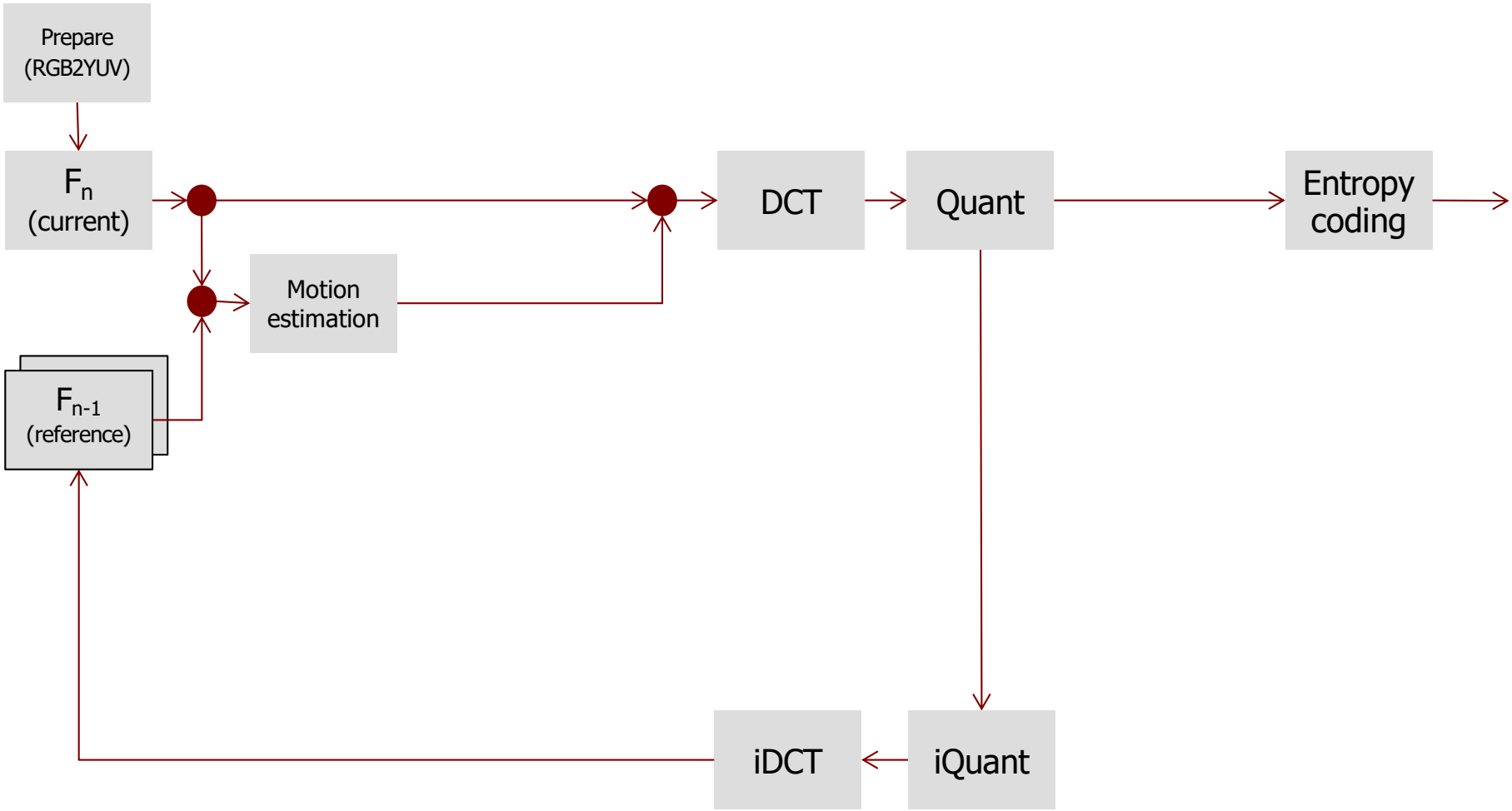
Motion Compensation

- 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

Adding Motion to the JPEG pipeline



Adding Motion to the JPEG pipeline



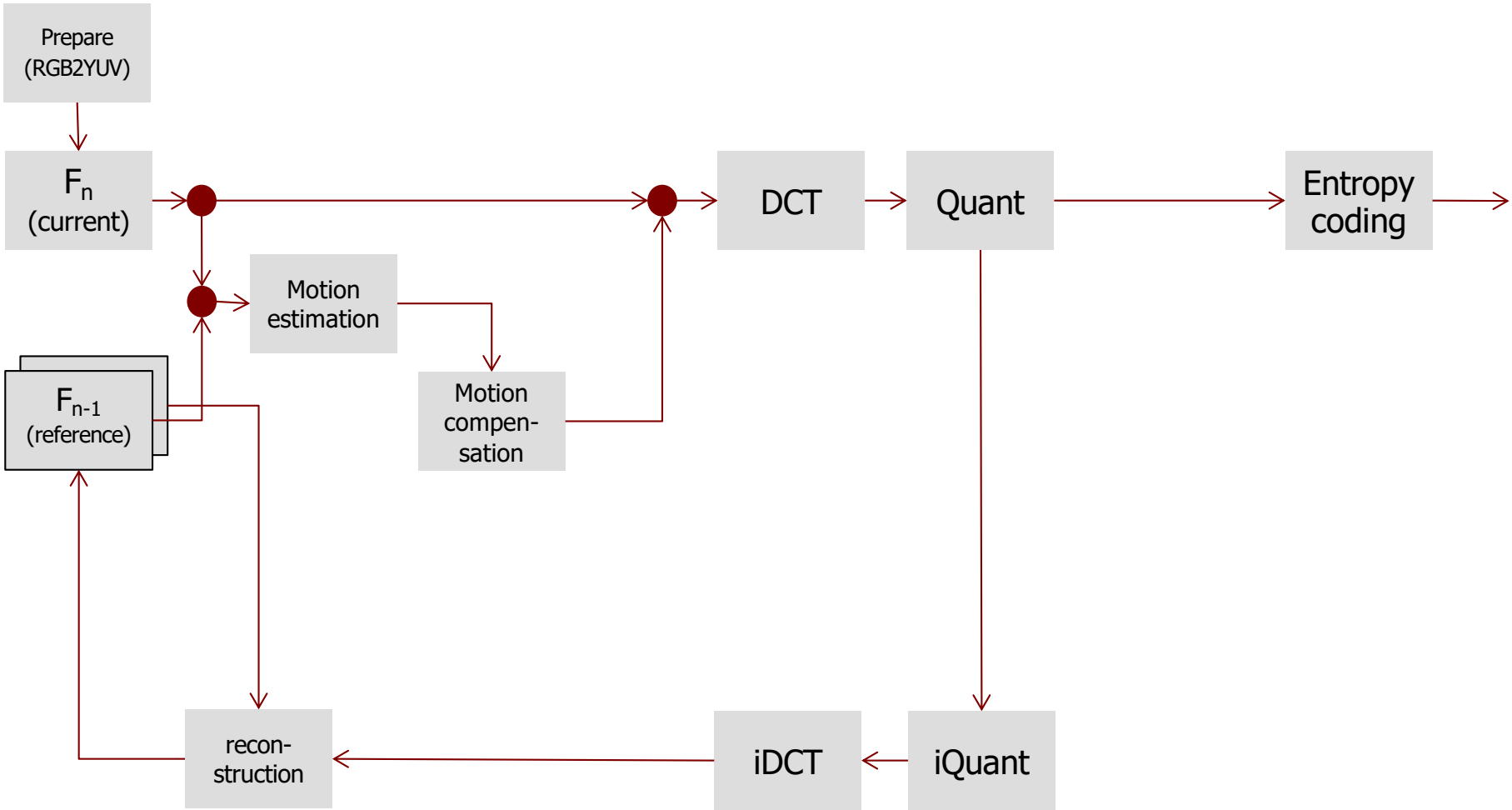
Frame Reconstruction

- 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 (roughly) the same *reconstructed* frame as the decoder will receive

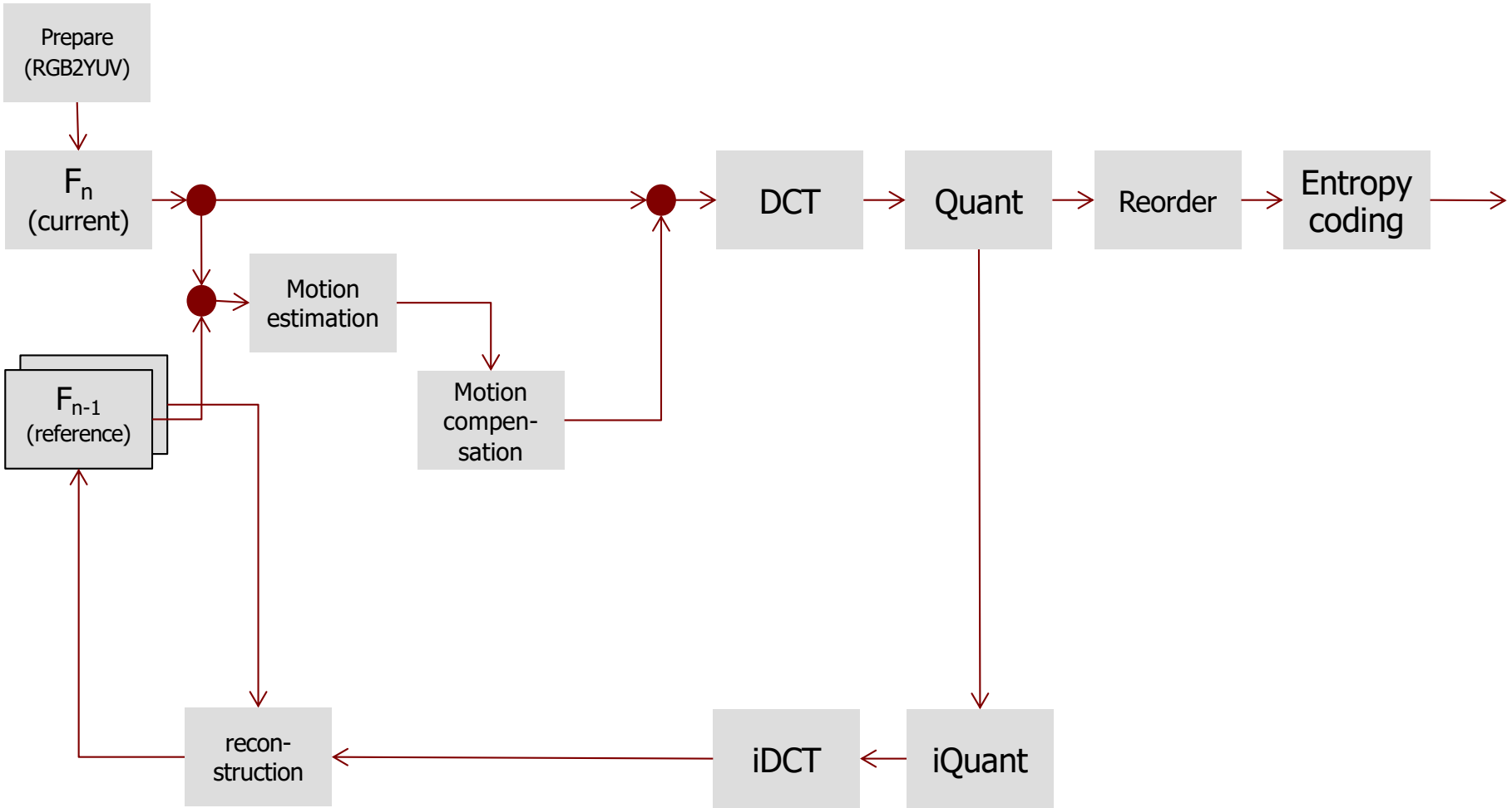
Residual Transformation

- The pixel difference between the original frame and the reconstructed frame 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 and Quantization
- MPEG uses special Quantization tables for residuals
- in IN5050, we don't

Adding Motion to the JPEG pipeline

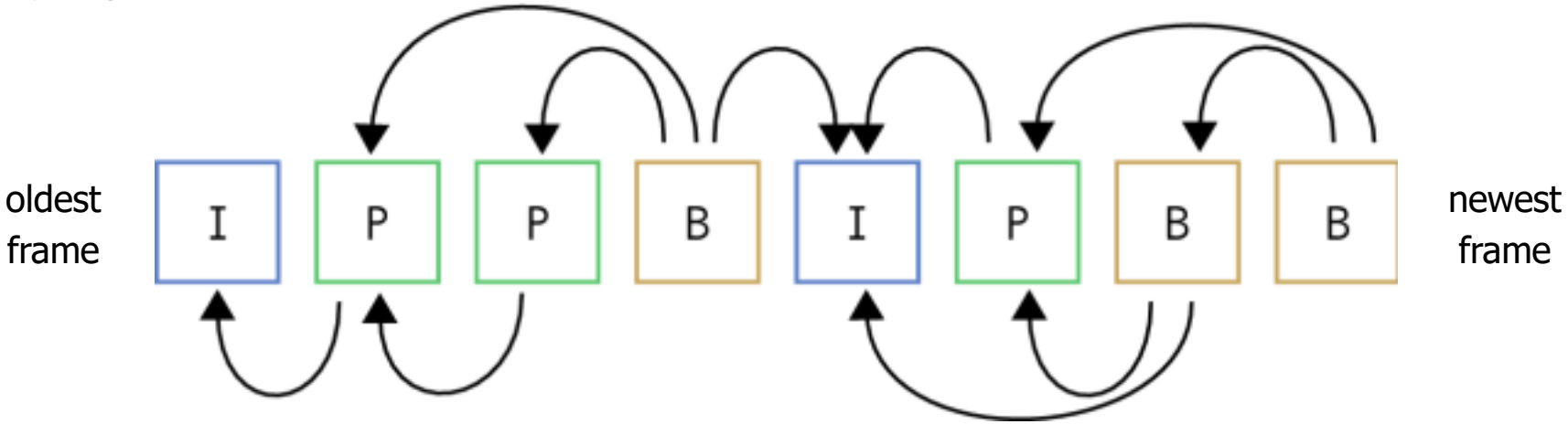


Adding Motion to the JPEG pipeline



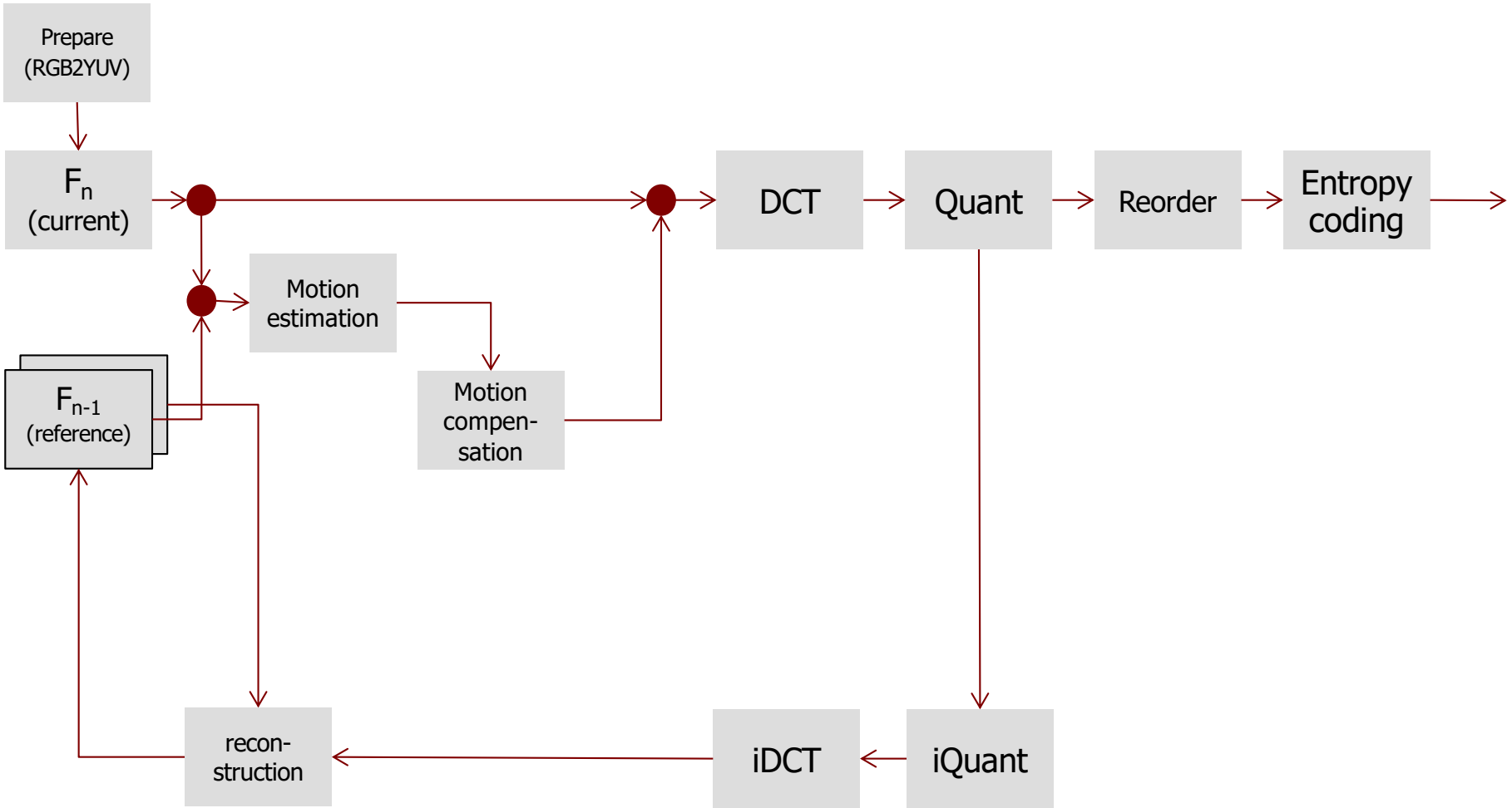
Macroblock Types

- MPEG-1 macroblocks are 8x8 pixels, they can be of type I, P or B
  - Intra-MBs use Intra-prediction (like JPEG)
  - Predicted MBs use Inter-prediction
  - Bi-Directional Predicted MBs use prediction both backward and forward in time

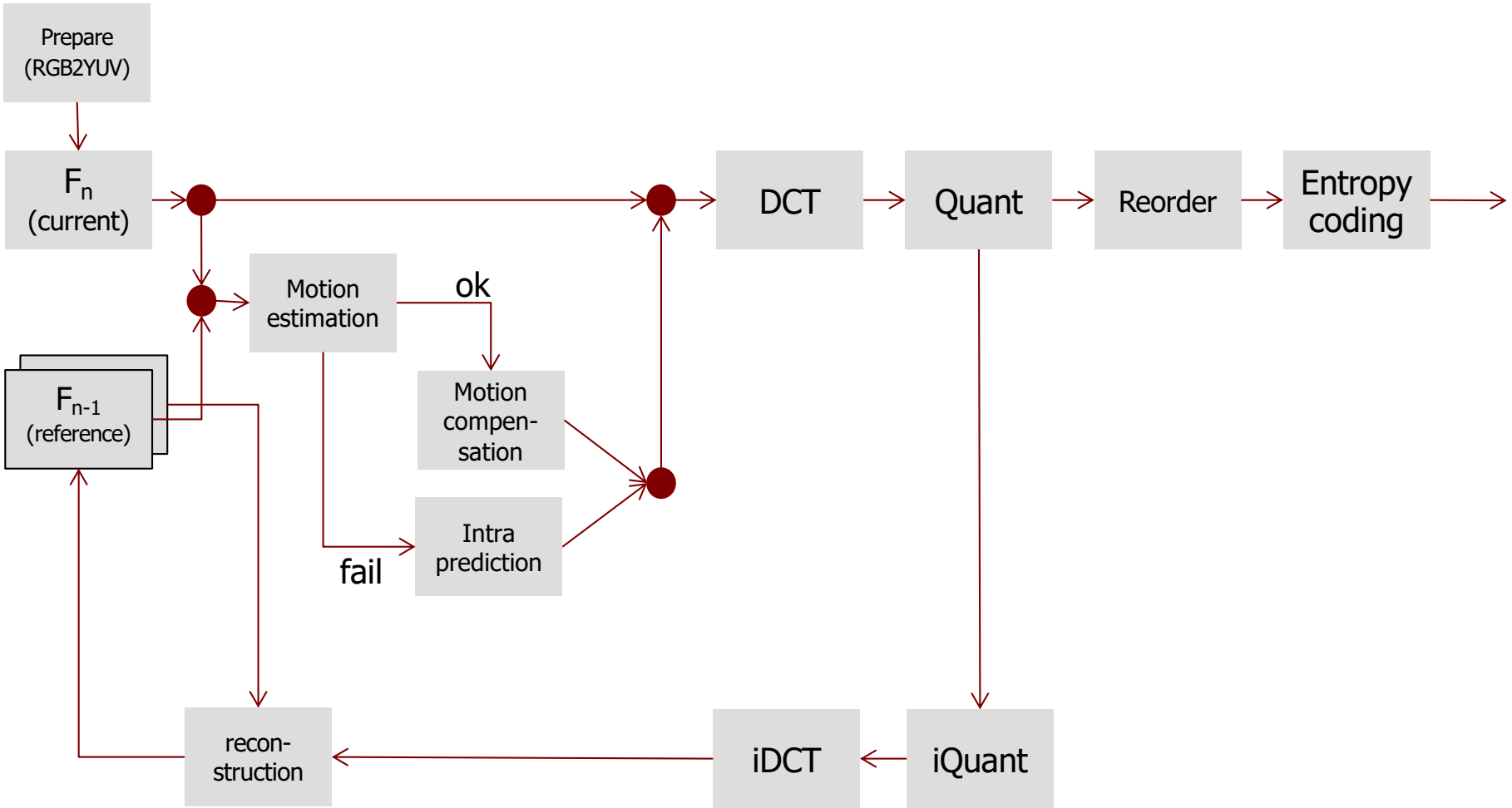


  - The frames are reordered in such a way that frames used for predicting other frames are decoded first

Adding Motion to the JPEG pipeline



Simplified MPEG Encoder Overview



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 (forgetting about all the failed tests) usually accounts for a small percentage of the running time
  - This makes ***de****coding* cheap
  - An important goal of MPEG

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