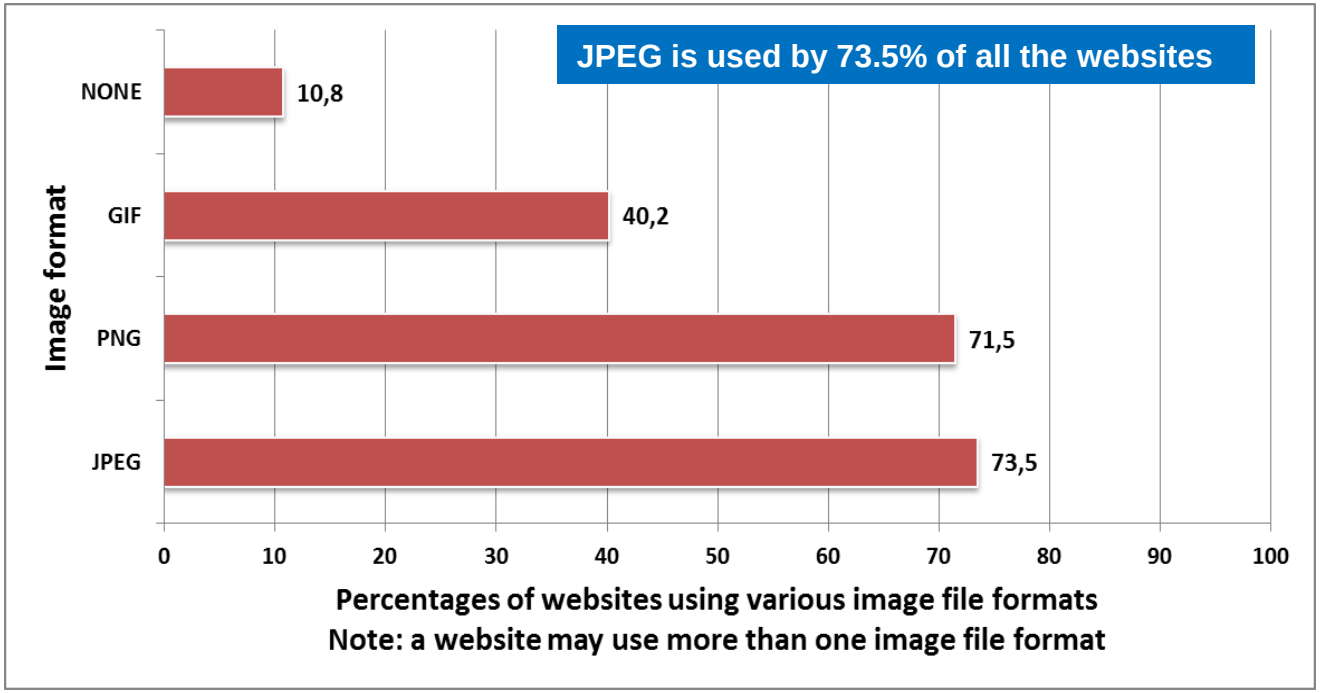
What is JPEG?

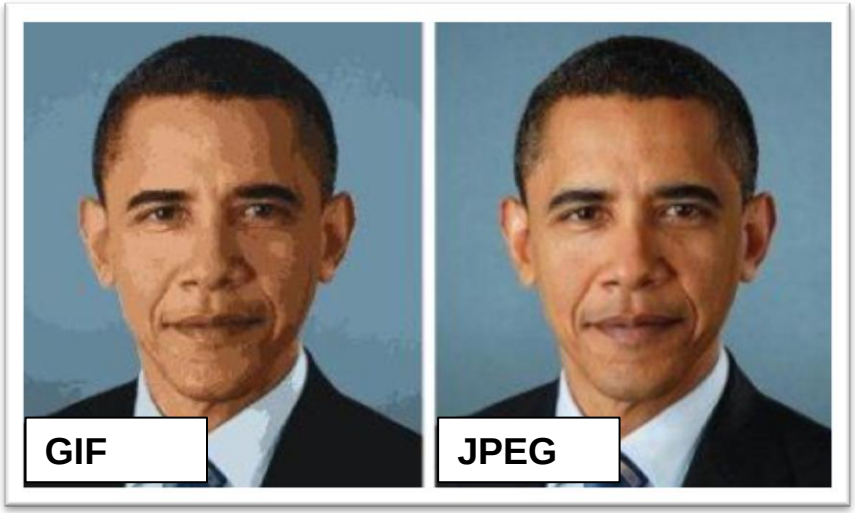
- JPEG (Joint Photographic Expert Group) is an international standard for **lossy image compression** released in 1992
- JPEG is still today one of the most popular image formats on the Web



Source: https://w3techs.com/technologies/overview/image_format/all (updated April 2016)

What is JPEG?

JPEG is used in many applications. It is particularly suitable for the compression of **photographs and paintings of realistic scenes** with smooth variations of tone and color



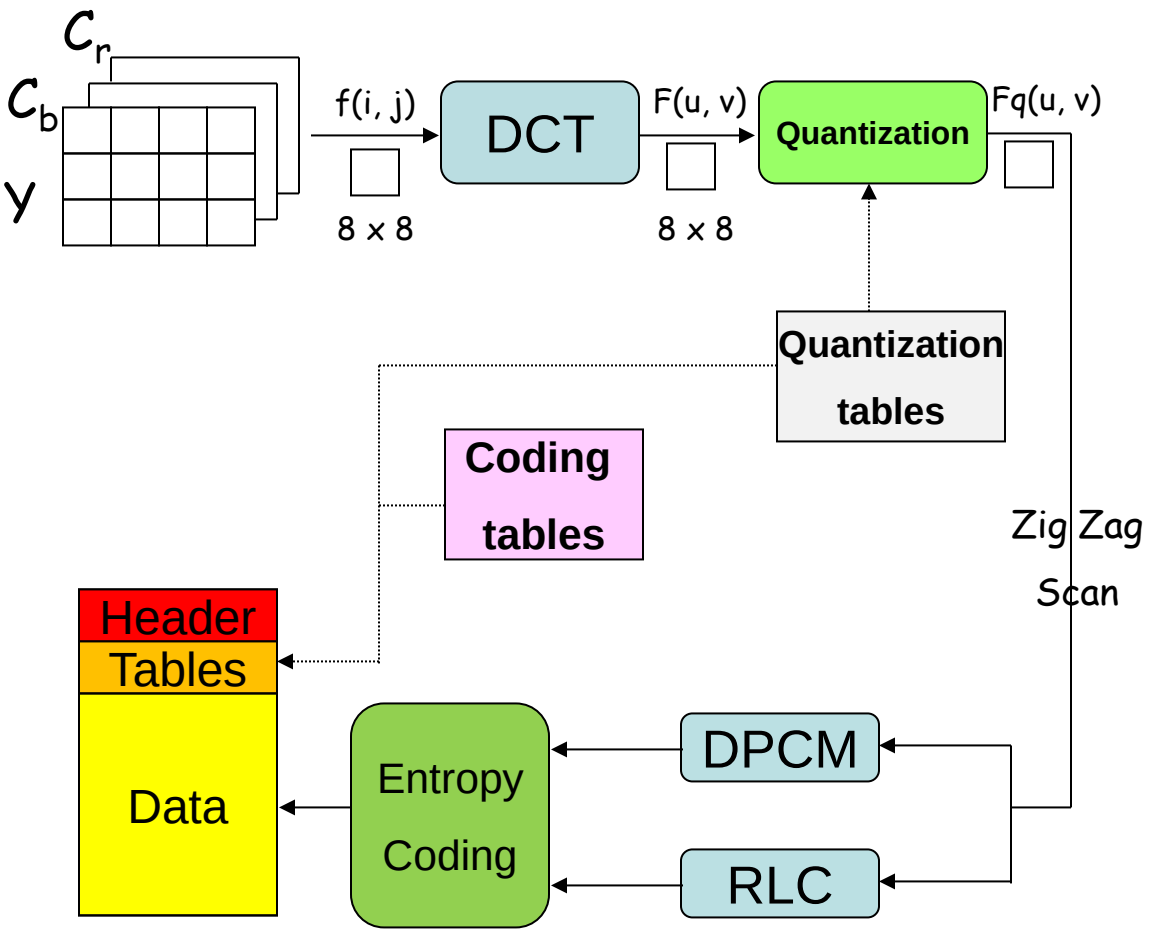
With respect to the also widely diffused GIF format, **JPEG ensures better visual quality** compressed images **for the same file size**

How does it work?

JPEG achieves good trade-off between *visual quality* and *compression efficiency* by exploiting a number of algorithms

- Color Space Transform and subsampling
- Discrete Cosine Transform (DCT)
- Quantization
- Zig-zag ordering
- Differential Pulse Code Modulation (DC component)
- Run Length Encoding (AC components)
- Entropy Coding (Huffman or Arithmetic)

JPEG baseline encoding



Main steps:

1. Discrete Cosine Transform of each 8x8 pixel block
2. Scalar quantization
3. Zig-zag scan to exploit redundancy
4. Differential Pulse Code Modulation (DPCM) on the DC component and Run Length Encoding of the AC components
5. Entropy coding (Huffman)

Reverse order for decoding

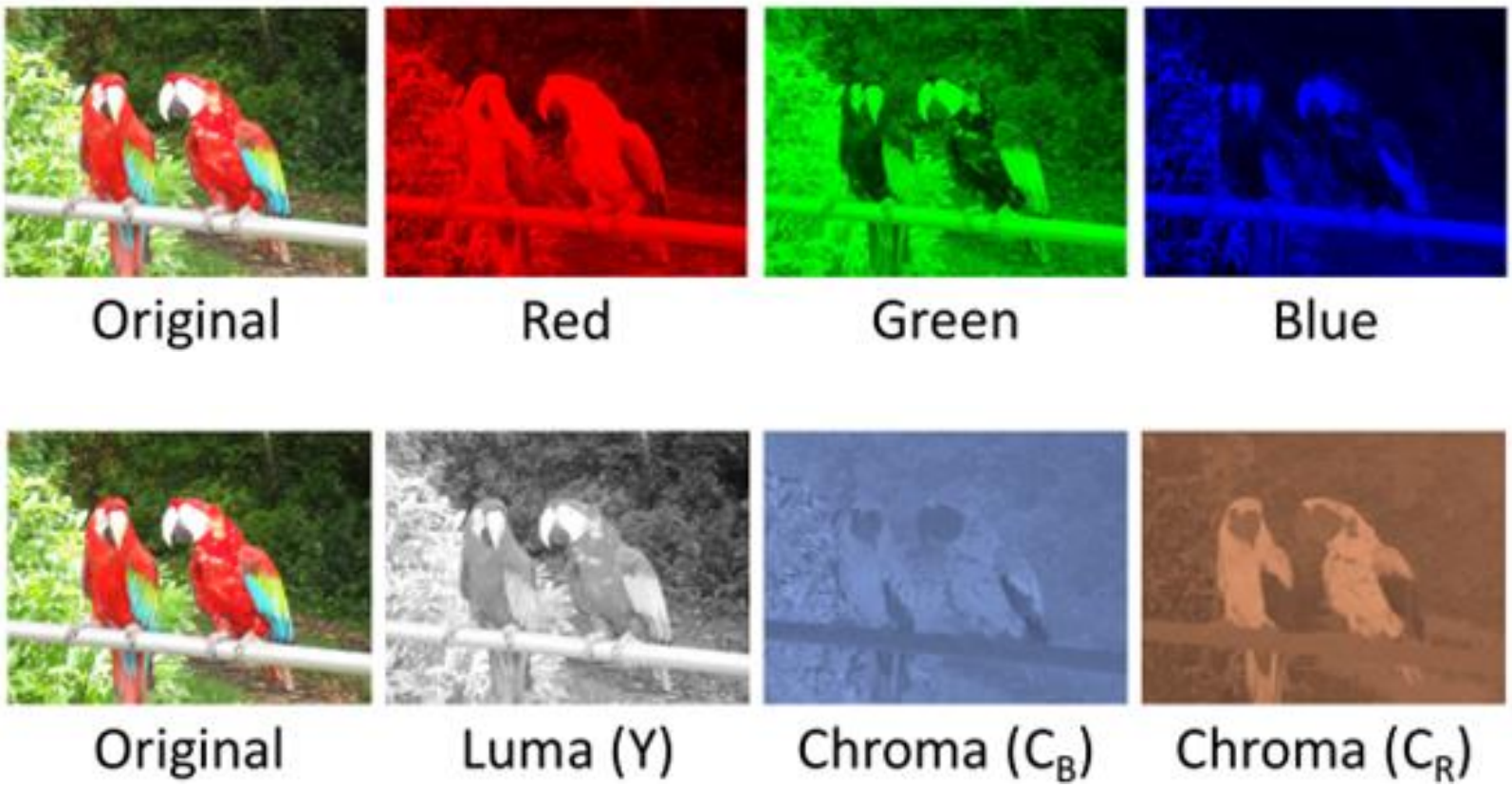
Color space transform: RGB to YCbCr

- **RGB** color space is not the only method to represent an image
- There are several other color spaces, each one with its properties
- A popular color space in image compression is the **YCbCr**, which:
  - separates luminance (Y) from color information (Cb,Cr)
  - processes Y and (Cb,Cr) separately (not possible in RGB)
- RGB to YCbCr and YCbCr to RGB linear conversions:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \quad \begin{matrix} Y \in [0, 255] \\ C_b \in [0, 255] \\ C_r \in [0, 255] \end{matrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.773 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{bmatrix} \quad \begin{matrix} R \in [0, 255] \\ G \in [0, 255] \\ B \in [0, 255] \end{matrix}$$

Color space transform – example

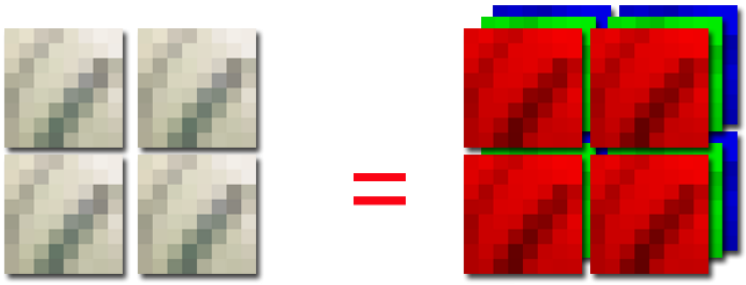


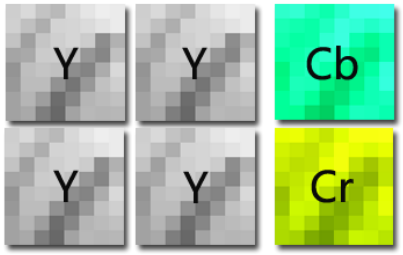
Color space transform – subsampling

- Y is taken every pixel, and Cb,Cr are taken for a block of 2x2 pixels

Data size is reduced to a half without significant losses in visual quality

- Example: block 64x64





Without subsampling, one must take 64^2 pixel values for each color channel:

$3 * 64^2 = 12288$ values (1 bytes per value)

JPEG takes 64^2 values for Y and $2x32^2$ values for chroma

$64^2 + 2x32^2 = 6144$ values (1 bytes per value)

Discrete Cosine Transform (DCT)

- **Transformed data are more suitable to compression** (e.g. skew probability distribution, reduced correlation).
- No lossy

Forward DCT

$$F(u,v) = \frac{1}{4} C(u)C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right]$$

$$\text{for } u = 0,...,7 \text{ and } v = 0,...,7$$

$$\text{where } C(k) = \begin{cases} 1/\sqrt{2} & \text{for } k = 0 \\ 1 & \text{otherwise} \end{cases}$$

Inverse DCT

$$f(x,y) = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C(u)C(v)F(u,v) \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right]$$

$$\text{for } x = 0,...,7 \text{ and } y = 0,...,7$$

Quantization

- **Goal: to reduce number of bits per sample**
- For each 8x8 DCT block, *F(u.v)* is divided by a 8x8 **quantization matrix *Q***

$$F_q(u,v) = \left\lfloor \frac{F(u,v)}{Q(u,v)} \right\rfloor, \quad \hat{F}(u,v) = F_q(u,v) \cdot Q(u,v)$$

$$Err(u,v) = \hat{F}(u,v) - F_q(u,v)$$

Q(u,v), *quantization step* at frequency (u,v)

- Example: F = 45 = $(101101)_2$ (6 bits)
  - Truncate to 4 bits (Q=4): $(1011)_2$ = 11. (De-quantize: 11x4 = 44 against 45)
  - Truncate to 3 bits (Q=8): $(101)_2$ = 5. (De-quantize: 8x5 = 40 against 45)
- **Quantization error is the main reason why JPEG compression is LOSSY**

Quantization

- Each *F[u,v]* in a 8x8 block is divided by constant value *Q(u,v)*
- Higher values in the quantization matrix Q achieve *better compression* at the cost of *visual quality*
- **How to choose Q?**
- Eye is more sensitive to low frequencies (upper left corner of the 8x8 block) less sensitive to high frequencies (lower right corner)
- **Idea: quantize more (large quantization step) the high frequencies, less the low frequencies**
- The values of Q are controlled with a parameter called **Quality Factor (QF) which ranges from 100 (best quality) to 1 (extremely low)**

Quantization: luminance

- Quantization table Q for **QF = 50**

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	36	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Quantization: chrominance

- Quantization table Q for **QF = 50**
- Can quantized color more coarsely due to reduced sensitivity of the Human Visual System (HVS)

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Quantization: luminance and chrominance

- An example of quantization table Q for **QF = 70**

Quantization Table: Luminance							
10	7	6	10	14	24	31	37
7	7	8	11	16	35	36	33
8	8	10	14	24	34	41	34
8	10	13	17	31	52	48	37
11	13	22	34	41	65	62	46
14	21	33	38	49	62	68	55
29	38	47	52	62	73	72	61
43	55	57	59	67	60	62	59

Quantization Table: Chrominance							
10	11	14	28	59	59	59	59
11	13	16	40	59	59	59	59
14	16	34	59	59	59	59	59
28	40	59	59	59	59	59	59
59	59	59	59	59	59	59	59
59	59	59	59	59	59	59	59
59	59	59	59	59	59	59	59
59	59	59	59	59	59	59	59

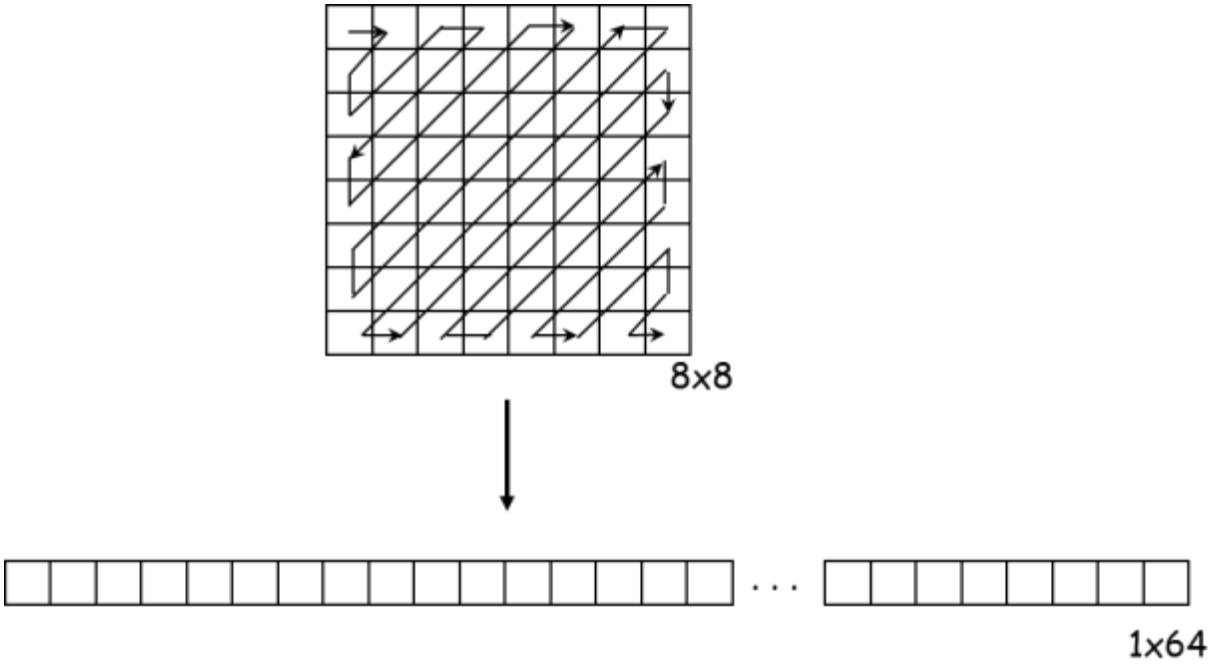
- The quantization is less strong at larger QF


NO JPEG (20MB)
JPEG 100 (9MB)
JPEG 60 (1.3MB)
JPEG 20 (0.6MB)
JPEG 5 (0.4MB)

Zig-zag ordering of quantized coefficients

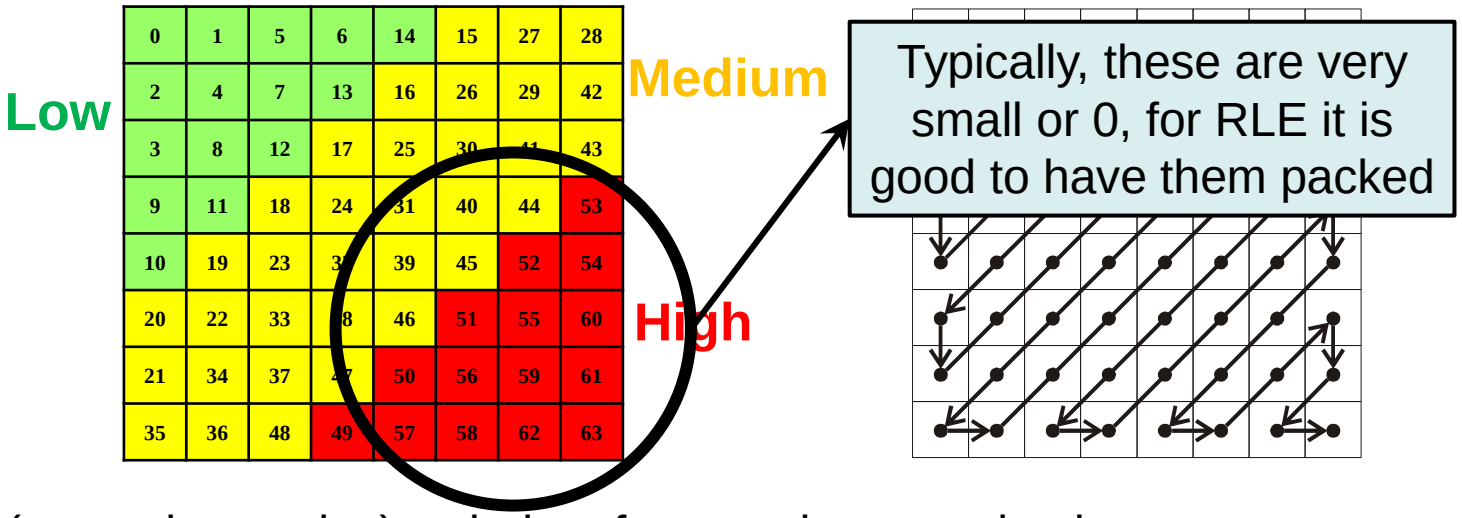
- For further processing, each 8x8 block is converted to a 1x64 vector

- To do so, **JPEG adopts a method called zig-zag scan**, which packs together low, medium and high frequency coefficients



Zig-zag ordering of quantized coefficients

- Packing coefficients in a clever way



- Normal (e.g. column-wise) ordering: frequencies are mixed

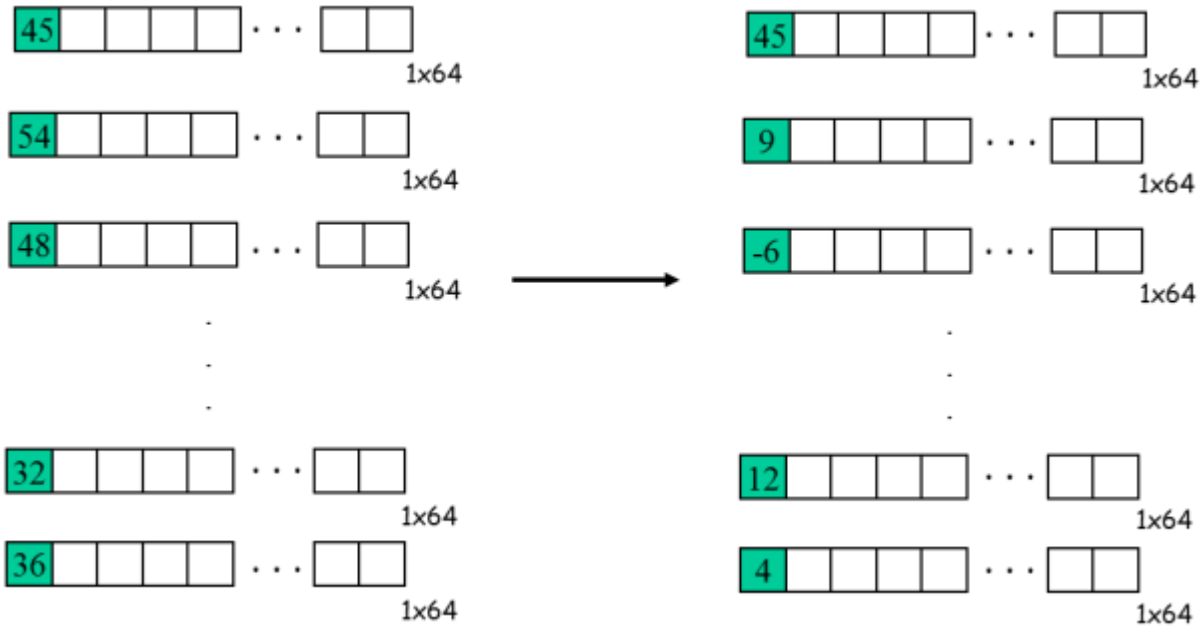
0	2	3	5	10	20	21	35	1	…	47	49	14	16	…	46	50	…	64

- Zig-zag: frequencies are better sorted

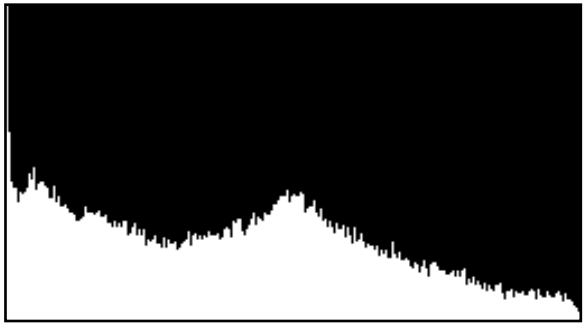
0	2	3	9	10	1	4	8	.....	20	19	18	.....	49	50	.....	60	61	63

DPCM on DC component

- DC component is large and can assume several different values. However, often **the difference between DCs of two adjacent blocks is not large**
- To save bits, encode the difference from DC of previous 8x8 blocks
  - This procedure is called Differential Pulse Code Modulation (DPCM)



DPCM on DC component



DPCM: Laplacian distribution sharply peaked in 0 (right)

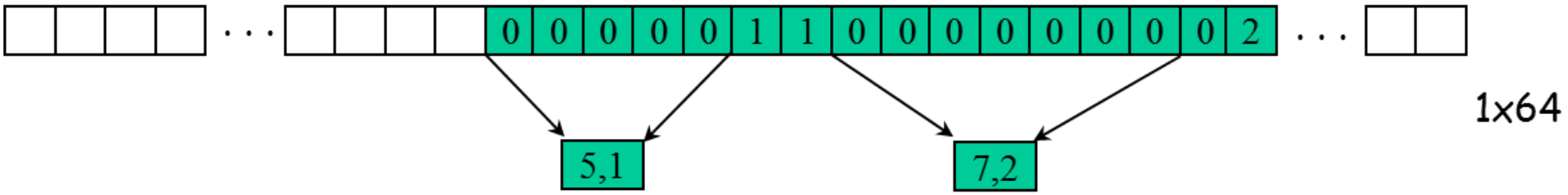
Pixels: uniform distribution in [0,255] (left)

The **entropy of the error image is much smaller** than that of the original image

RLE on AC components

- The **1x64 vectors have a lot of zeros**, more so towards the end of the vector.
  - Higher up entries in the vector capture higher frequency (DCT) components which tend to be capture less of the content.

- Encode a series of 0s as a (*skip*,*value*) pair, where *skip* is the number of zeros and *value* is the next non-zero component.
  - Send (0,0) as end-of-block sentinel value.



Entropy coding

- DC components are differentially coded as (SIZE,Value)
  - The code for a **Value** is derived from the following table

SIZE	Value	Code
0	0	---
1	-1,1	0,1
2	-3, -2, 2,3	00,01,10,11
3	-7,…, -4, 4,…, 7	000,…, 011, 100,…111
4	-15,…, -8, 8,…, 15	0000,…, 0111, 1000,…, 1111
.		.
.		.
11	-2047,…, -1024, 1024,… 2047	…

Entropy coding

- **DC components are differentially coded as (SIZE,Value)**. The code for a SIZE is derived from the following table

SIZE	Length	Code
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

Example: If a DC component is 40 and the previous DC component is 48. The difference is -8. Therefore it is coded as:

1010111

0111: The value for representing –8 (see Size_and_Value table)

101: The size from the same table reads 4. The corresponding code from the table at left is 101.