Chapter 7
End-to-End Data

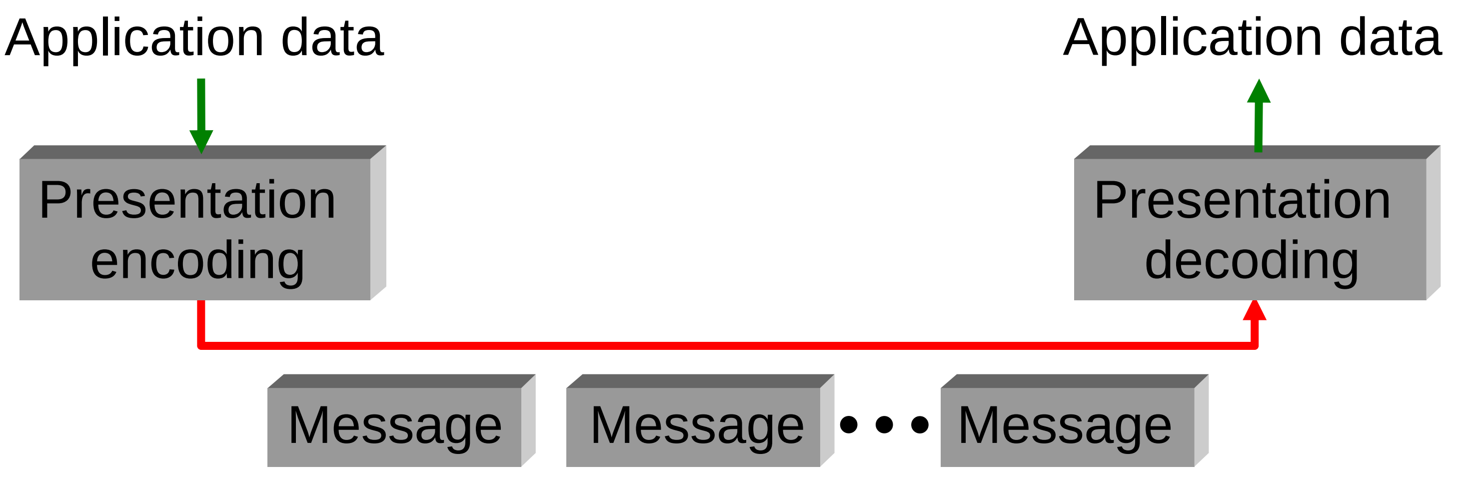
End-to-End Data

- Discuss the problem of how to **best encode** different kinds of data that application programs want to exchange
  - Let the receiver be able to extract **the same message** from the signal as the transmitter sent
    - The two sides agreeing to a **message format**, called the **presentation format**
- To make the encoding as **efficient** as possible
  - In one hand, **add** as much **redundancy** in the data as possible ⇒ The receiver can extract the **right data** even if errors are introduced into the message **(error correction)**
  - In the other hand, **remove** as much **redundancy** from the data as possible ⇒ the message is encoded **as few bits as possible** **(data compression)**

Presentation Formatting

Presentation Formatting

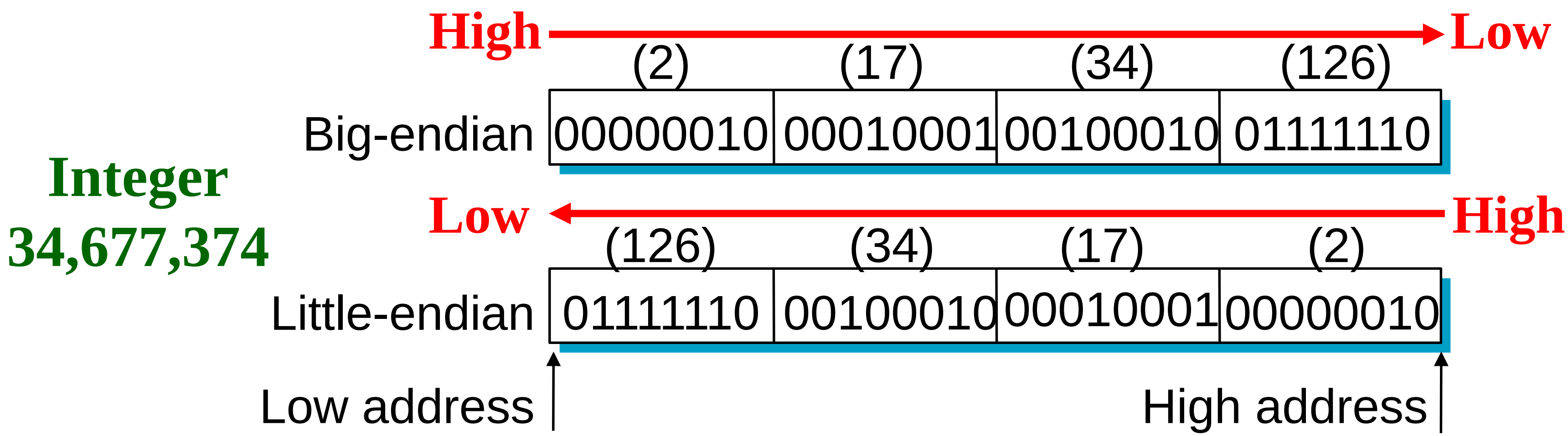
- **Encoding:** the sender translates the data from the **representation** it uses internally into a **message** that can be transmitted over the network **(argument marshalling)**
  - Such as "Image ⇒ Message" or "Voice ⇒ Message"
- **Decoding:** the receiver translates the arriving **message** into a **representation** that it can then process **(unmarshalling)**
  - Such as "Message ⇒ Image" or "Message ⇒ Voice"



Presentation Formatting (Problems)

- Computers represent data in **different ways**
  - **Big-endian form** versus **little-endian form**
- Application programs are written in **different languages**
  - Even when they are using the same language, there may have more than one **complier**

⇒ We **cannot** simply transmit a structure from one machine to another

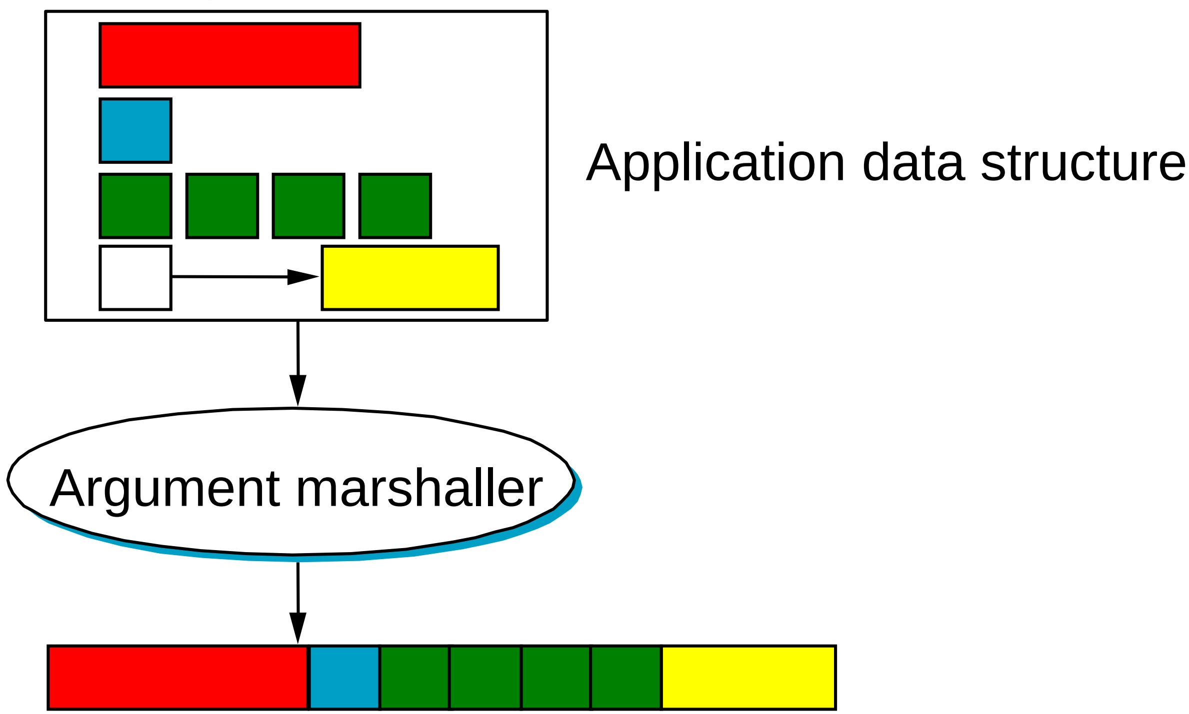


Data Types

- The data type system includes three levels
- **The lowest level: base types**, including **integers, floating-point numbers** and **characters**; might also support **ordinal types** and **booleans**
  - Converts each base type from one representation to another (such as from big-endian to little-endian)
- **The next level: flat types**, including **structures** and **arrays**
  - The compiler sometimes insert **padding** between fields
  - The marshalling system packs structures with **no padding**
- **The highest level: complex types**, built using **pointers**
  - The data might not be contained in a single structure (involves pointer from one structure to another)

Data Types

- The task of argument marshalling usually involves
  - **Converting** the base types,
  - **Packing** the structures, and
  - **Linearizing** the complex data structures



Conversion Strategy

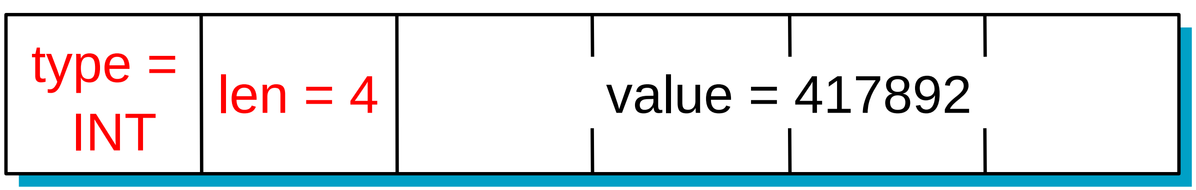
- There are two general options of conversion strategy
- **Canonical intermediate form:**
  - The **sender** translates from its **internal** representation to an **external** representation before sending data
  - The **receiver** translates from this **external** representation into its **local** representation when receiving data
- **Receiver-makes-right:**
  - The **sender** transmit data in its own **internal** format
  - The **receiver** is responsible for translating the data from the **sender's format** into its **local format**
  - Every host must be prepared to convert data from **all other** machine architectures

Conversion Strategy

- **Receiver-makes-right** is an ***N*-by-*N* solution**
  - Each of *N* machine architectures must be able to handle **all *N* architectures**
- For **canonical intermediate form**, each host needs to know only how to convert between **its own** representation and the **external** one
- Is the canonical intermediate form the best choice?
  - The number of machine architectures ***N* is not so large**
  - The most common case is for two machines of **the same type** to be communicating with each other
- A third option is to use **receiver-makes-right** if the sender and destination has **the same architecture**, and use **canonical intermediate form** if they are **different**

Tags

- How to let the receiver know **what kind of data** is contained in the message?
  - Two approaches: **tagged** and **untagged** data
- A tag is any **additional information** included in a message
  - **Type tag:** indicates that the value is an integer, a floating-point number, or whatever
  - **Length tag:** indicates the number of elements in an array or the size of an integer
  - **Architecture tag:** is used in conjunction with the **receiver-makes-right** strategy to specify the architecture



Tags

- The alternative is **not to use tags**
  - It knows because it was **programmed to know**
  - If you **call a remote procedure** that takes two integers and a floating-point number as argument
    - The remote procedure does not need to inspect tags to know what has just received
    - It simply **assumes** that the message contains two integers and a floating-point number
- The untagged data works for most cases
  - Only **breaks down** for sending **variable-length arrays**
    - A **length tag** is commonly used

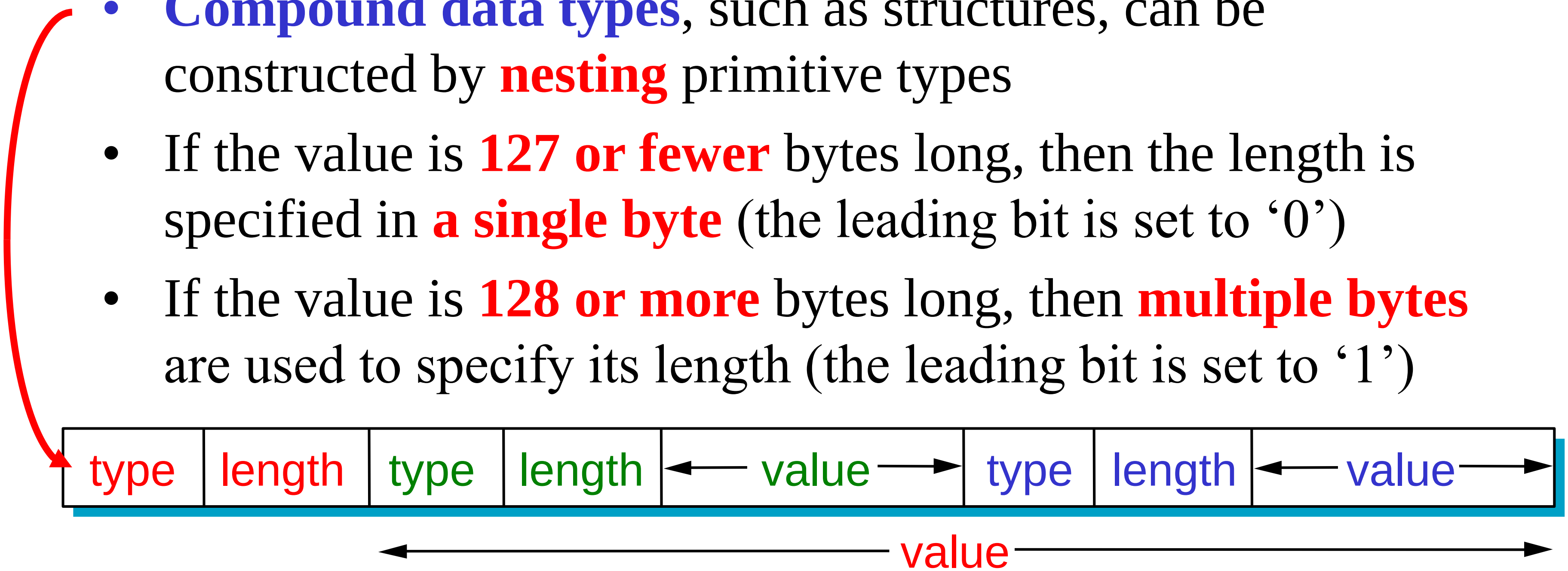
ASN.1 (Abstract Syntax Notation One)

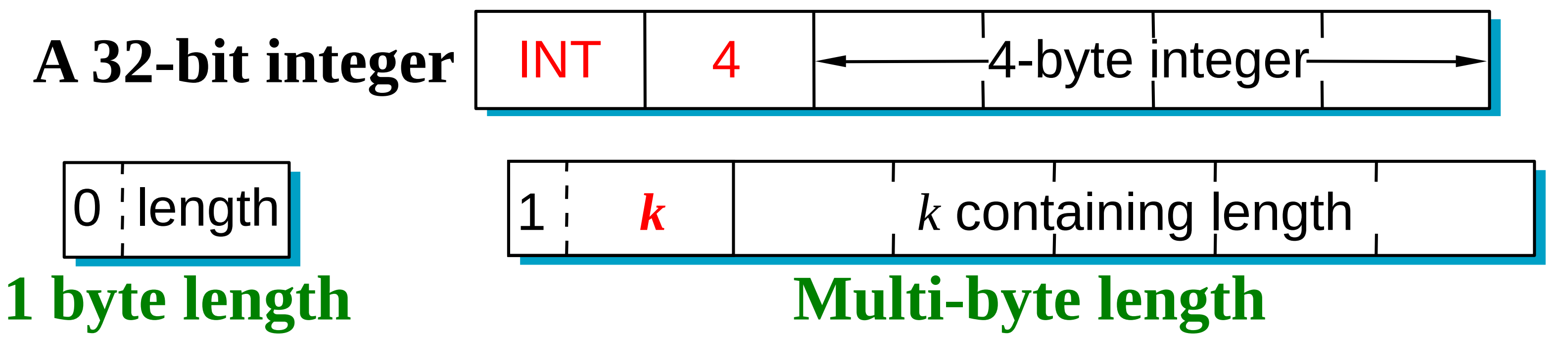
- ASN.1 is an **ISO standard** that defines a representation for data sent over a network
  - Support the entire **C type system** (except function pointers)
  - Define a **canonical intermediate form**
  - Uses type tags
- The representation-specific part is called the **Basic Encoding Rules (BER)**
- ASN.1 represents each data item with a triple of the form

  <tag, length, value>
  - The tag is typically an 8-bit field
  - The length field specifies the length, in bytes, of the value

ASN.1 (Abstract Syntax Notation One)

- **Compound data types**, such as structures, can be constructed by **nesting** primitive types
- If the value is **127 or fewer** bytes long, then the length is specified in **a single byte** (the leading bit is set to '0')
- If the value is **128 or more** bytes long, then **multiple bytes** are used to specify its length (the leading bit is set to '1')





Data Compression

Compression

- How many bits do you need to represent a stream of binary digits or a stream of alphabets?
  - 110011011110001111000110000011111….
  - Aabsndkjs dsjfjfdfjfjk fsdkja fas dsjfs aff …
- Entropy: the average number of bits needed for each symbol.
- Information theory: to find the fundamental limit
- Coding theory: to find ways to achieve the fundamental limit

Suppose that $X_1, X_2, \ldots, X_n, \ldots$ are i.i.d. random variables.

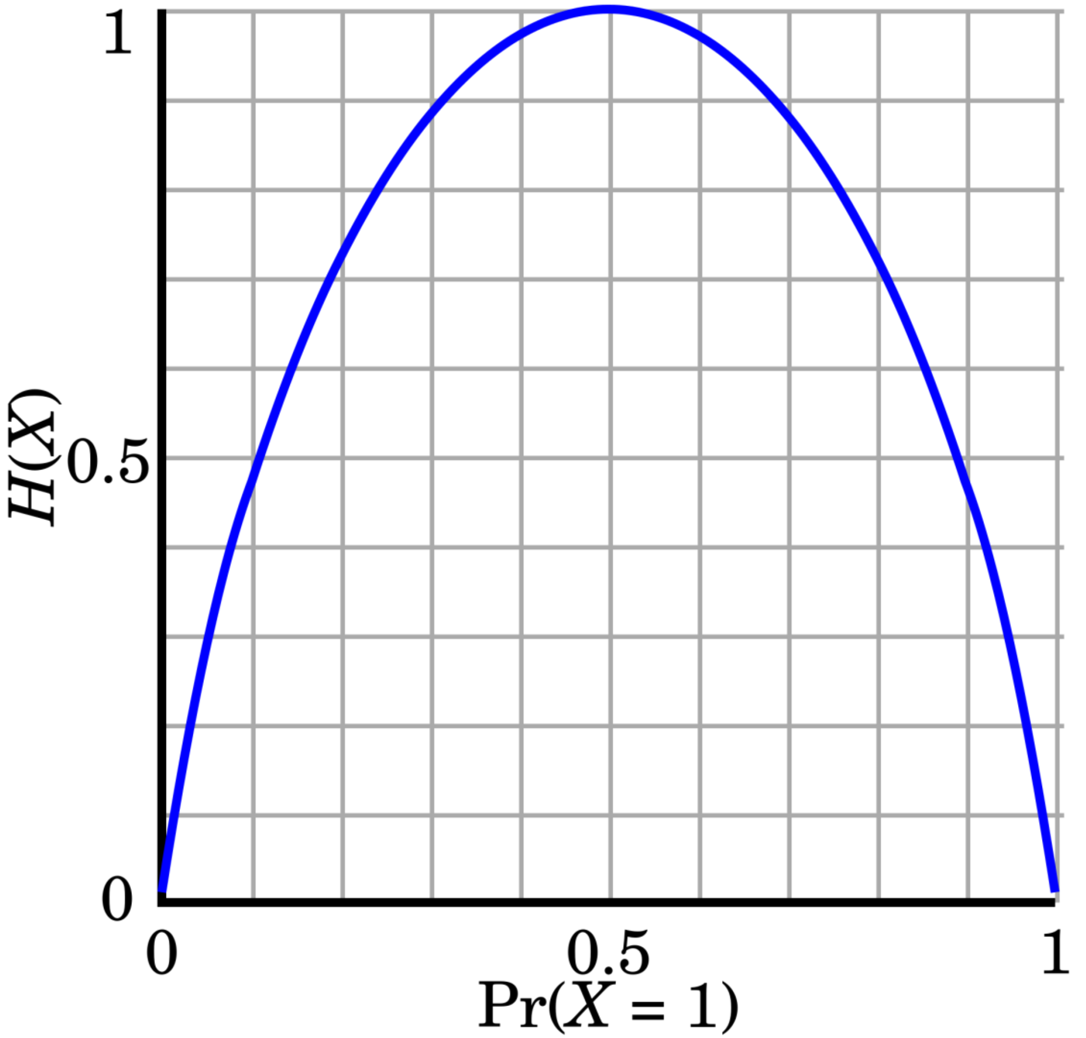
There are a set of symbols $\mathcal{X}$, $S_0, S_1, \ldots S_k, \ldots$ with

$$P(X_1 = S_k) = p_k.$$

The entropy is defined as

$$H(X) = -\sum_{k \in \mathcal{X}} p_k \log p_k.$$

Entropy for Bernoulli random variables



Data Compression

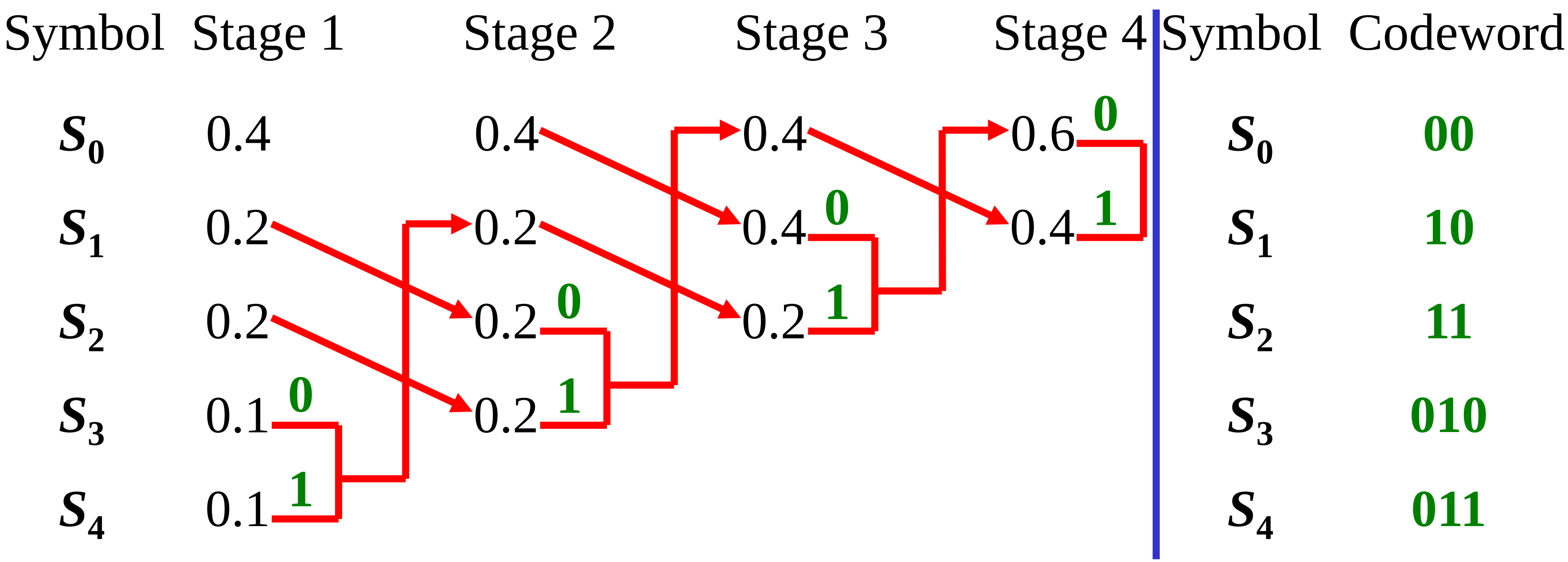
- Sometimes application programs need to send **more** data in a timely fashion than the bandwidth of the network supports
  - A **10-Mbps** video stream wants to transmit over a network with **1-Mbps** available bandwidth
  - First **compress** the data at the sender, then
  - Transmit it over the network, and
  - Finally to **decompress** it at the receiver
- **Compression** is inseparable from **data encoding**
  - The **Huffman codes**
    - Encode the data according to the relative probability of each symbol

Huffman encoding algorithm

- First one parses through the stream of alphabets to find the probability of each symbol.
- Combine the two least probable source symbols into a new single symbol, whose probability is equal to the sum of the probabilities of the original two. Thus we have to encode a new source alphabet of one less symbol. Repeating this step until we get down to the problem of encoding just two symbols in a source alphabet, which can be encoded merely using 0 and 1.
- Go backward by splitting one of the two (combined) symbols into two original symbols, and the codewords of the two split symbols is formed by appending 0 for one of them and 1 for the other from the codeword of their combined symbol. Repeating this step until all the original symbols have been recovered and obtained a codeword.

Huffman Coding

- A **high** probability symbol is assigned a **short** codeword
- A **low** probability symbol is assigned a **long** codeword



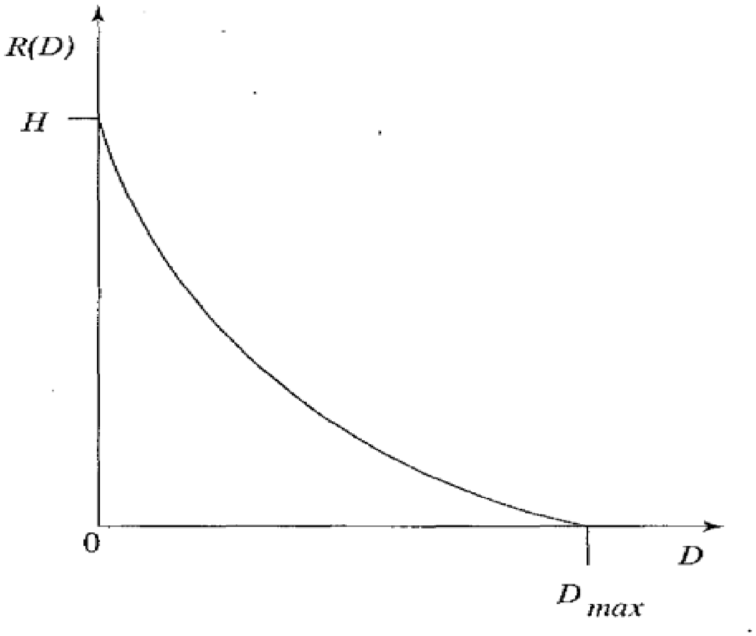
The tree of the Huffman encoding

Data Compression

- There are two classes of compression algorithms
  - **Lossless compression:** ensures that the data recovered from the compression/decompression process is **exactly the same** as the original data
    - Used to compress **file data**, such as executable codes, text files and numeric data
  - **Lossy compression:** **does not promise** that the data received is exactly the same as the data sent
    - Removes information that it cannot later be restored
    - The lost information **will not miss the reception**
    - Used to compress **still images**, **video**, and **audio**
- The **lossy** algorithms achieve much better compression ratios

The rate-distortion theory

- Lossy compression always involves a tradeoff between rate and distortion.
- Rate is the average number of bits required to represent each source symbol.
- Within this framework, the tradeoff between rate and distortion is represented in the form of a rate-distortion function R(D).



Data Compression

- Compression/decompression algorithms often involve **time-consuming computations**
- Is the compression beneficial?
  - $\boldsymbol{B_c}$**:** denotes the average bandwidth at which data can be pushed through the compressor and decompressor
  - $\boldsymbol{B_n}$**:** denotes the network bandwidth (including network processing costs) for uncompressed data
  - $\boldsymbol{r}$**:** denotes the average compression ratio
  - The time taken to send $\boldsymbol{x}$ bytes of uncompressed data is $\boldsymbol{x / B_n}$

Data Compression

- The time to compress it and send the compressed data is
  - compression time + transmission time
  - $x / B_c + x / (r B_n)$
- The compression is beneficial if

  $x/B_c + x / (r B_n) < x/B_n$

  $\Rightarrow B_c > r / (r - 1) \times B_n$
- If $r = 2$, $B_c > 2 \times B_n$

Lossless Compression

Lossless Compression Algorithms (RLE)

- **Run Length Encoding (RLE)** is a compression technique with a **brute-force** simplicity
  - Replace **consecutive occurrences** of a given symbol with
    - one copy of the symbol, plus a count of the repetition
  - **AAABBCDDDD → 3A2B1C4D**
- RLE can be used to compress digital image by comparing adjacent pixel values and then **encoding only the changes**
  - Same adjacent pixel values ⇒ the encoding value is ‘0’
  - Large homogeneous regions ⇒ a large amount of consecutive ‘0’
  - For images having large **homogeneous regions** ⇒ **effective**

Lossless Compression Algorithms (RLE)

- For **text images**, they contain a large amount of **white space** that can be removed
  - RLE is a key algorithm used to transmit **faxes**
- RLE is **not effective** for images with a **small degree** of local variation
  - It may takes **2 bytes** to represent a single symbol when that symbol is not repeated

Lossless Compression Algorithms (DPCM)

- **Differential Pulse Code Modulation (DPCM):**
  - First output a **reference symbol** and then,
  - For each symbol, to output **the difference** between that symbol and the reference symbol
  - AAABBCDDDD → takes **A** as the reference
    - **A0001123333**
- When the differences are **small**, they can be encoded with **fewer bits** than the symbol itself

Lossless Compression Algorithms (DPCM)

- DPCM works better than RLE for most **digital image**
  - The dynamic range of **"the differences between adjacent pixel values"** is significantly **less** than that of the original image
  - A compression ratios of **1.5-to-1** can be obtained on digital images
- Another approach is **delta encoding:**
  - AAABBCDDDD → **A001011000**
- It is possible to perform RLE after delta encoding
  - Since the output generally has consecutive occurrences of a given symbol

Lossless Compression Algorithms (DB)

- For **Dictionary-Based (DB)** methods, the **Lempel-Ziv (LZ) compression algorithm** is the best known
  - **Build a dictionary (table)** of variable-length strings that are expected to find in the data
  - **Replace** each of these string when it appears in the data with the **corresponding index** to the dictionary
- For example, **"compression"** has the index **4978** in one particular dictionary
  - "compression" would be replaced by 4978
  - "compression" requires **77 bits** for encoding by **7-bit ASCII**
  - If the dictionary has 25,000 words → it takes just **15 bits**

Lossless Compression Algorithms (DB)

- To find the dictionary, a solution is to **adaptively** define the dictionary **based on the contents** of the data being compressed
  - The constructed dictionary has to be sent along with the data

- Variation of LZ used to compress GIF images
  - first reduce 24-bit color to 8-bit color
  - treat common sequence of pixels as terms in dictionary
  - not uncommon to achieve 10-to-1 compression (x3)

Lempel-Ziv Codes

Algorithm:

Parse the input sequence into strings that have never appeared before.

For example.

The input sequence is 1011010100010………;

Step 1:

- **The algorithm first parses the first letter 1 and finds that it never appears before . So 1 is the first string .**
- **Then the algorithm parses the second letter 0 and finds that it never appears before. Thus, the algorithm puts it to be the next string .**
- **The algorithm parses the next letter 1, and finds that this string has appeared. Hence, it parses another letter 1 and yields a new string 11.**
- **Repeat these procedures. The source sequence is parsed into strings as**

  1 ; 0 ; 11 ; 01 ; 010 ; 00 ; 10 ………..

- Step 2:

L = 8. So the indices will be:

parsed source : 1 0 11 01 010 00 10 ….

index : 001 010 011 100 101 110 111

E.g.

the codeword of source string 010 will be the index of 01, (i.e. 100), concatenated with the last bit of the source string, (i.e. 0).

- The codeword string is:

(000 , 1)(000 , 0)(001 , 1)(010 , 1)(100 , 0)(010 , 0)(001 , 0)

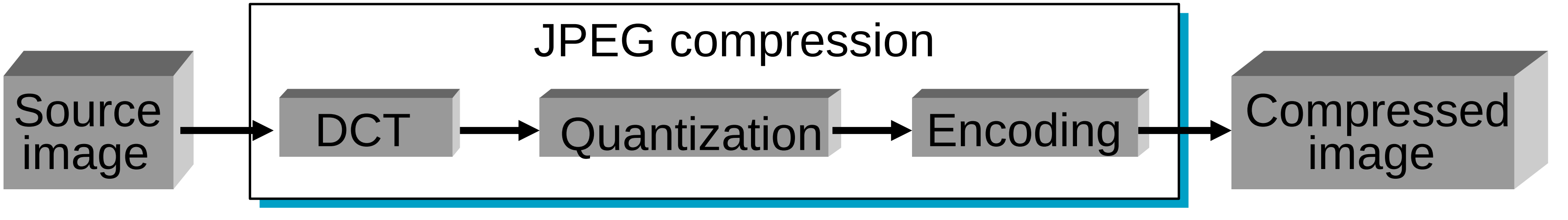
or equivalently,

0001 0000 0011 0101 1000 0100 0010 ……..

Lossy Compression

Image Compression (JPEG)

- **JPEG (Joint Photographic Experts Group):** more than just a compression algorithm
  - It also define the **format** for images
- JPEG compression takes place in three phases:
  - **DCT (Discrete Cosine Transform):** transforms the signal into an equivalent signal in the **spatial frequency domain**
  - **Quantization:** **loses the least significant information** contained in that signal
  - **Encoding:** adds an element of **lossless compression** to the lossy compression achieved by the first two phases



Fourier Series Formula

cuemath
THE MATH EXPERT

$$f(x) = \frac{1}{2}a_0 + \Sigma_{n=1}^{\infty} a_n \cos nx + \Sigma_{n=1}^{\infty} b_n \sin nx$$

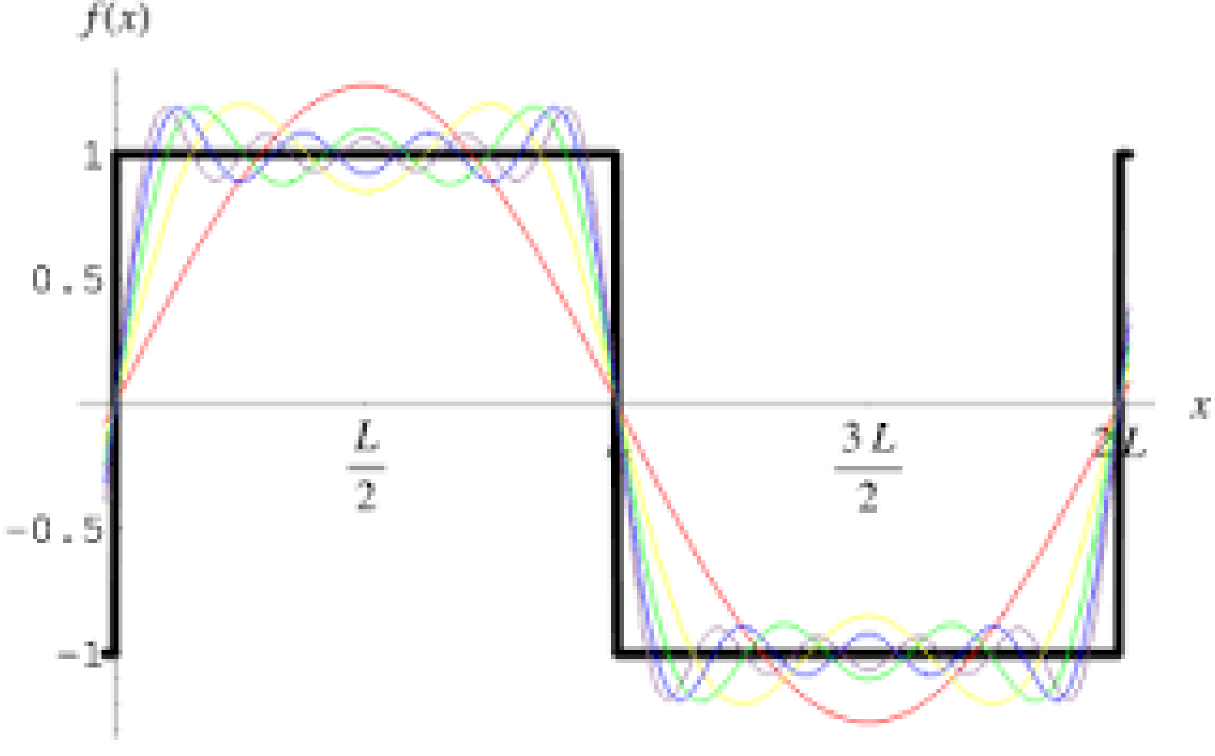
where,

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x)\, dx$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos nx\, dx$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin nx\, dx$$

$n = 1,2,3 \ldots..$


f(x)
1
0.5
-0.5
-1
x
L/2
3L/2
2L

1D Quantization

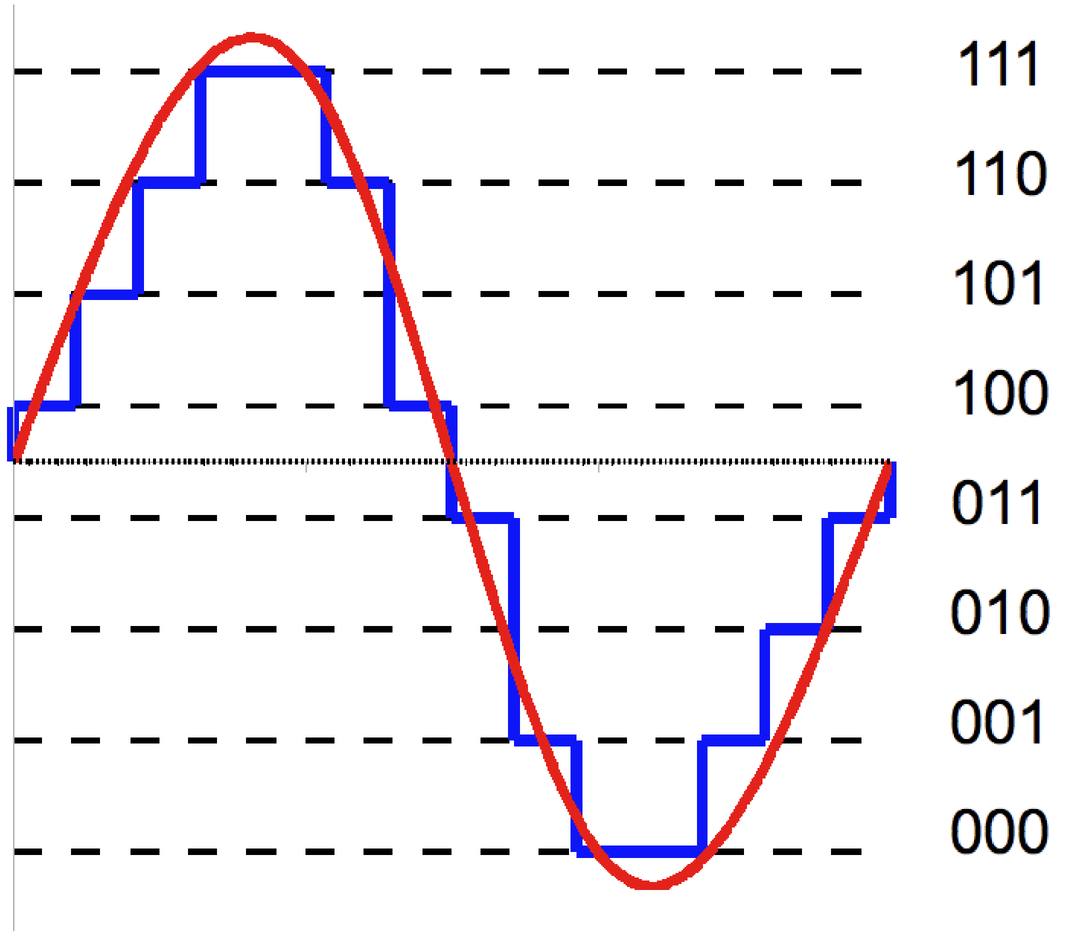


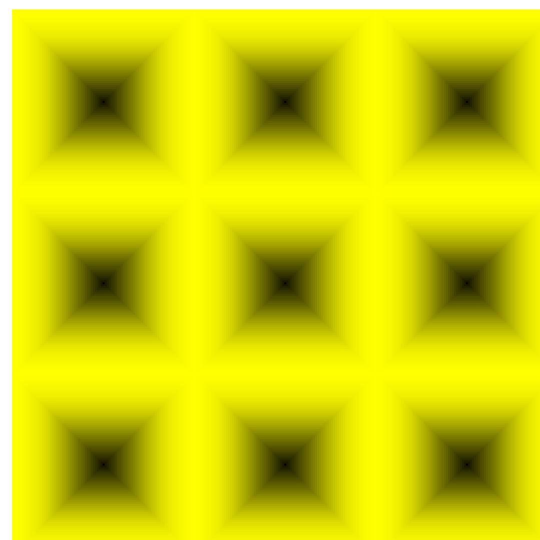
Image Compression (JPEG)

- DCT takes an **8 × 8** matrix of **pixel values** as input and output an **8 × 8** matrix of **frequency coefficients**
- If the value **changes slowly**, it has a **low spatial frequency**; and if it changes **rapidly**, it has a **high spatial frequency**
  - The low frequencies correspond to the **gross features**
  - The high frequencies correspond to the **fine detail**
  - The gross features are **essential** and the fine detail is **less essential**
- Moving from low-frequency information to high-frequency information, the image becomes finer and finer detail
- The high-frequency coefficients are increasingly **unimportant** to the perceived quality of the image

Image Compression (JPEG)

- Low spatial frequency: the value changes slowly

- High spatial frequency: the value changes rapidly



DCT (Discrete Cosine Transform)

$$DCT(i,j) = \frac{1}{\sqrt{2N}} C(i)C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} pixel(x,y) \cos[\frac{(2x+1)i\pi}{2N}] \cos[\frac{(2y+1)j\pi}{2N}]$$

$$pixel(i,j) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i)C(j) DCT(i,j) \cos[\frac{(2x+1)i\pi}{2N}] \cos[\frac{(2y+1)j\pi}{2N}]$$

$$C(x) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

Fourier Cosine Transforms

$$\hat{f}_c(\omega) = \sqrt{\frac{2}{\pi}} \int_0^\infty f(x) \cos \omega x \, dx$$

$$f(x) = \sqrt{\frac{2}{\pi}} \int_0^\infty \hat{f}(\omega) \cos \omega x \, d\omega$$

Quantization equation

$$QuantizedValue(i,j) = IntegerRound(DCT(i,j)/Quantum(i,j))$$

where

$$IntegerRound(x) = \begin{cases} \lfloor x + 0.5 \rfloor & \text{if} \quad x \geq 0 \\ \lfloor x - 0.5 \rfloor & \text{if} \quad x < 0 \end{cases}$$

Decompression

$$DCT(i,j) = QuantizedValue(i,j) \times Quantum(i,j)$$

Image Compression (JPEG)

- **DCT does not lose information:** just transforms the image into another form for information removing
- **Quantization:** a matter of **dropping the insignificant bits** of the frequency coefficients
- The **low coefficients** have a **quantum close to 1**
  - Little low-frequency information is lost
- The **high coefficients** have **larger values** of quantums
  - Many high coefficients end up being set to **0** after quantization

Quantization Table
Quantum step
8 × 8

$$Quantum = \begin{bmatrix} 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \\ 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 \\ 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 \\ 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 \\ 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 \\ 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \\ 15 & 17 & 19 & 21 & 23 & 25 & 27 & 29 \\ 17 & 19 & 21 & 23 & 25 & 27 & 29 & 31 \end{bmatrix}$$

Image Compression (JPEG)

- The final phase of JEPG encodes the quantized frequency coefficients in a compact form → **a lossless compression**
- Starting with the DC coefficient in **position (0, 0)**, the coefficients are processed in the zigzag sequence
- RLE is applied to only the 0 coefficients
  - **Many of the later coefficients are 0**
- The coefficient values are encoded using a **Huffman code**
- A image contains a large number of 8 × 8 blocks
  - Each **DC coefficient** is encoded as the **difference** from the previous DC coefficient

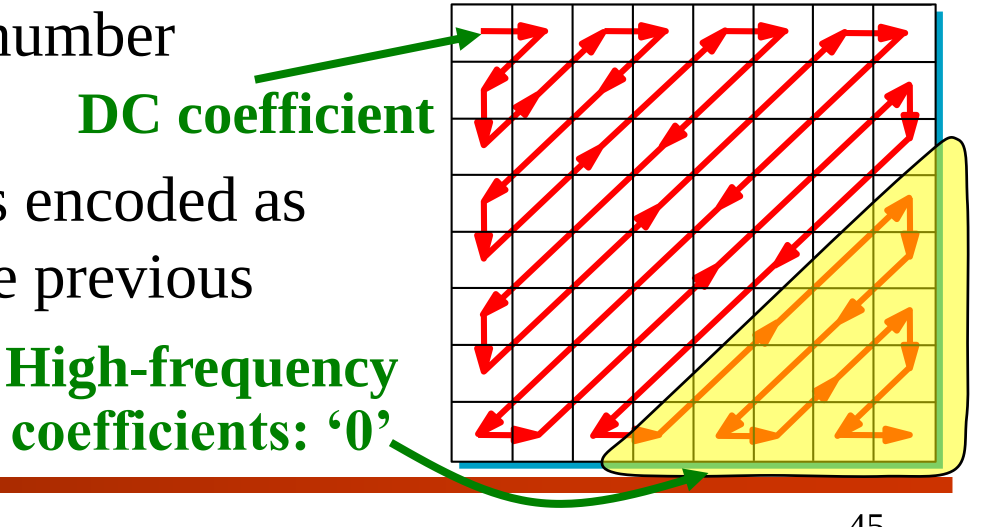


Image Compression (JPEG)

- For a **color image**, there are many different representations for each pixel to choose from
  - **RGB:** represents each pixel with **three color components**
    - **Red, Green and Blue**
  - **YUV:** has three components: one luminance (brightness) (Y) and two chrominance (U and V)

Lena

Original; 49KB

Compression rate 9, 6KB

Lena

Original; 49KB

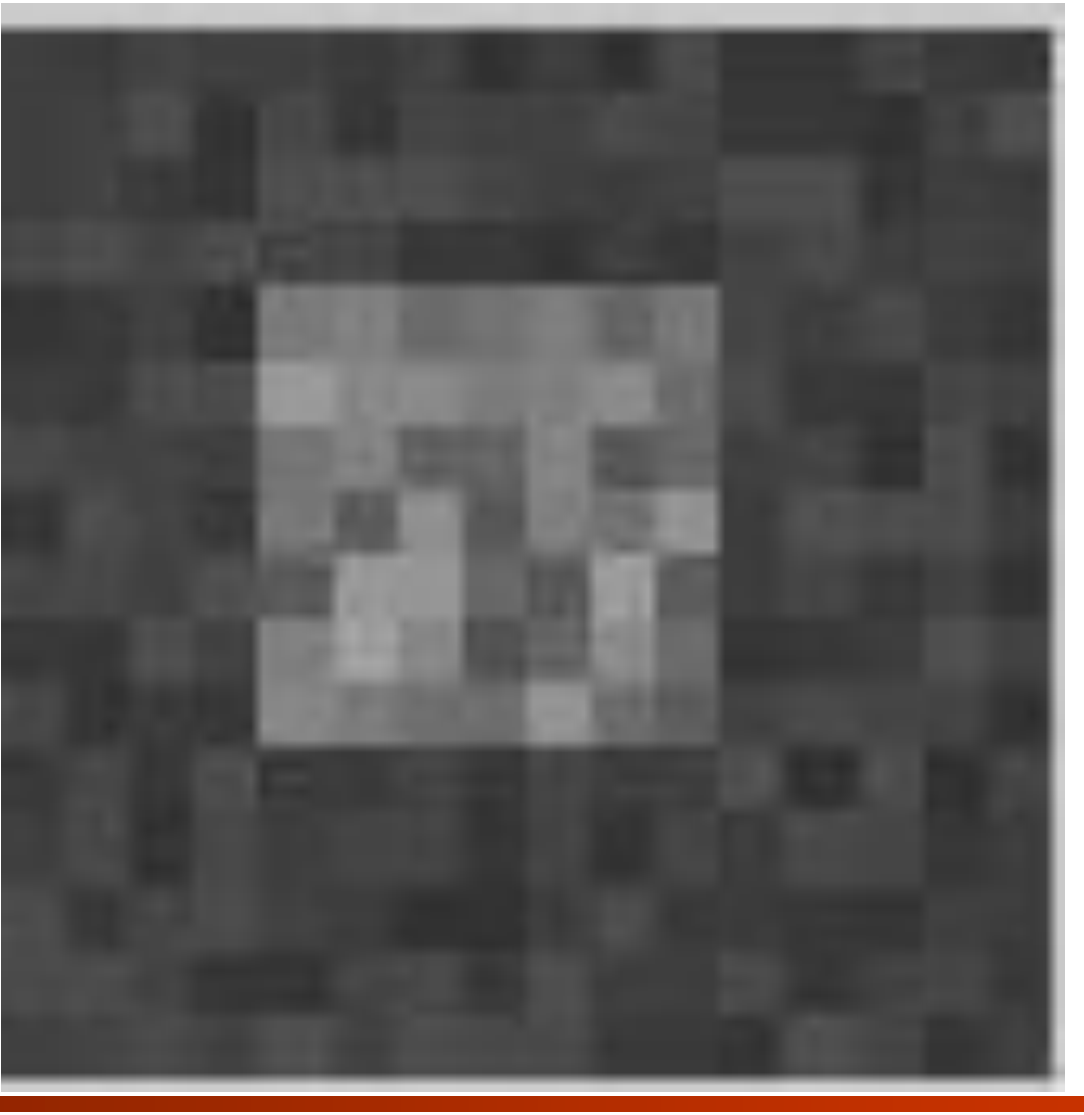
Compression rate 12, 4KB

Lena

Original; 49KB

Compression rate 20, 3KB

Original Image



Pixel Values of Original Image

63	60	69	64	64	59	66	50	63	51	75	54	54	69	59	56
62	66	79	52	72	54	69	66	65	75	71	56	55	51	65	80
62	67	59	53	80	79	79	69	65	62	66	78	78	54	63	60
72	78	69	77	70	70	56	58	52	63	55	68	74	62	66	63
53	58	66	51	123	129	108	115	127	102	121	71	63	74	61	54
61	59	77	76	154	136	138	124	128	140	109	76	58	55	70	66
73	68	68	64	118	129	99	100	133	89	104	65	69	50	77	52
54	77	68	57	120	94	143	89	128	114	147	60	80	76	76	63
70	72	65	80	91	152	150	103	88	156	93	62	69	57	66	52
57	56	79	67	128	159	135	83	101	139	107	53	57	62	79	67
72	53	60	56	135	123	111	110	147	109	94	66	74	71	68	50
58	72	56	77	60	60	69	56	70	62	63	80	50	77	50	69
64	73	53	72	63	69	69	54	68	52	69	60	74	71	55	62
76	58	70	76	65	60	50	51	62	77	58	60	56	54	64	60
75	76	68	55	54	69	74	58	79	63	63	74	56	67	69	58
64	69	75	66	80	66	77	73	75	59	57	54	80	71	54	65

Image Matrix after DCT Transformation

-391	-92	-21	47	-16	-30	-6	41	-426	95	32	-11	-20	7	14	32
-98	81	10	-36	6	18	8	-17	-87	-82	-28	-4	19	-6	-9	-25
-18	15	5	-8	-10	16	-18	14	3	1	9	-2	-12	3	18	19
39	-38	-14	11	10	-6	19	5	13	35	25	1	-11	6	6	-9
-20	14	1	-7	-6	7	-11	-5	2	-7	-17	8	-16	-2	25	5
-18	39	-5	3	9	7	9	-27	-36	-28	-17	15	17	11	-13	0
18	-25	-6	7	-2	7	-17	11	15	11	-8	-4	-6	-7	-4	-7
-2	-23	7	17	-5	-12	12	-3	6	17	-8	11	5	18	19	12
-412	-66	-18	52	-25	4	-4	6	-455	74	30	11	-20	-15	-4	-2
92	-92	-17	51	-23	-1	6	2	61	68	29	-14	-39	-21	-19	2
58	-43	3	17	-27	5	9	8	22	25	13	-4	-22	-23	-25	8
-13	23	6	-4	-11	22	1	-25	-19	-10	-8	-22	-17	-13	-24	-4
-18	14	10	-22	-4	11	-21	-22	-22	-25	-24	-1	11	-15	8	-8
-14	13	24	-11	0	6	-7	-12	-3	4	-6	1	7	15	-3	-21
11	-9	3	-2	26	-6	7	8	5	19	4	14	7	-8	-8	7
17	-20	16	14	0	-10	-2	4	17	17	-2	20	8	7	7	17

Quantization Matrix

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

Image Matrix after Quantization

-24	-8	-2	3	-1	-1	0	1	-27	9	3	-1	-1	0	0	1
-8	7	1	-2	0	0	0	0	-7	-7	-2	0	1	0	0	0
-1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
3	-2	-1	0	0	0	0	0	1	2	1	0	0	0	0	0
-1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	1	0	0	0	0	0	0	-2	-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	0	0	0	0	0	0	0
-26	-6	-2	3	-1	0	0	0	-28	7	3	1	-1	0	0	0
8	-8	-1	3	-1	0	0	0	5	6	2	-1	-2	0	0	0
4	-3	0	1	-1	0	0	0	2	2	1	0	-1	0	0	0
-1	1	0	0	0	0	0	0	-1	-1	0	-1	0	0	0	0
-1	1	0	0	0	0	0	0	-1	-1	-1	0	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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Image Matrix after Elimination

-24	-8	-2	3	-1	0	0	0	-27	9	3	-1	-1	0	0	0
-8	7	1	-2	0	0	0	0	-7	-7	-2	0	0	0	0	0
-1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0
3	-2	0	0	0	0	0	0	1	2	1	0	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	-2	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	0	0	0	0	0	0	0	0
-26	-6	-2	3	0	0	0	0	-28	7	3	1	0	0	0	0
8	-8	-1	3	0	0	0	0	5	6	2	0	0	0	0	0
4	-3	0	0	0	0	0	0	2	2	0	0	0	0	0	0
-1	1	0	0	0	0	0	0	-1	0	0	0	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
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	0

Image Matrix after Reconstruction (DCT)

-384	-88	-20	48	-24	0	0	0	-432	99	30	-16	-24	0	0	0
-96	84	14	-38	0	0	0	0	-84	-84	-28	0	0	0	0	0
-14	13	0	0	0	0	0	0	0	0	16	0	0	0	0	0
42	-34	0	0	0	0	0	0	14	34	22	0	0	0	0	0
-18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	-48	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	0	0	0	0	0	0	0	0

-416	-66	-20	48	0	0	0	0	-448	77	30	16	0	0	0	0
96	-96	-14	57	0	0	0	0	60	72	28	0	0	0	0	0
56	-39	0	0	0	0	0	0	28	26	0	0	0	0	0	0
-14	17	0	0	0	0	0	0	-14	0	0	0	0	0	0	0
-18	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
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	0

Image Matrix after Reconstruction

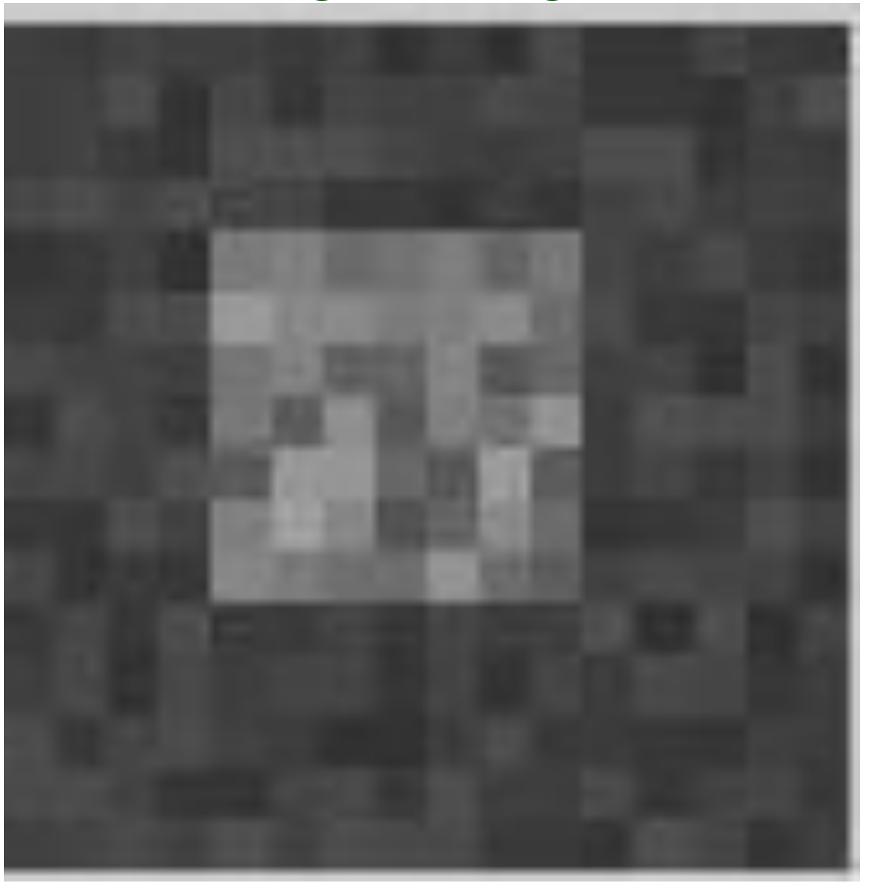
62	69	69	63	61	67	67	62	62	66	62	50	45	52	58	58
67	72	71	64	63	67	65	57	62	72	74	68	65	71	73	70
68	71	69	64	66	71	67	56	49	61	68	65	63	65	64	58
63	65	65	67	78	89	84	71	63	72	74	66	59	59	57	51
62	64	67	77	99	117	114	99	111	113	103	83	70	69	68	64
67	68	71	87	117	139	134	116	130	126	108	80	62	61	63	61
69	66	68	84	115	137	128	107	122	118	99	72	54	53	56	55
66	61	60	74	104	124	111	87	127	125	110	85	70	70	72	71
70	57	56	84	124	145	133	112	135	120	98	79	68	62	59	57
72	60	59	84	121	139	128	109	126	113	93	77	68	64	62	61
70	60	58	77	105	119	109	93	110	99	84	72	67	66	65	64
65	57	54	65	81	87	78	66	90	82	72	65	64	65	65	64
66	61	58	60	65	64	56	48	74	68	62	60	61	63	63	61
71	70	68	66	63	58	54	51	67	63	59	59	62	63	62	60
69	72	74	71	65	61	62	64	68	65	63	64	66	67	64	61
63	69	73	71	65	63	68	73	71	69	67	68	71	71	67	63

Compression Error Matrix

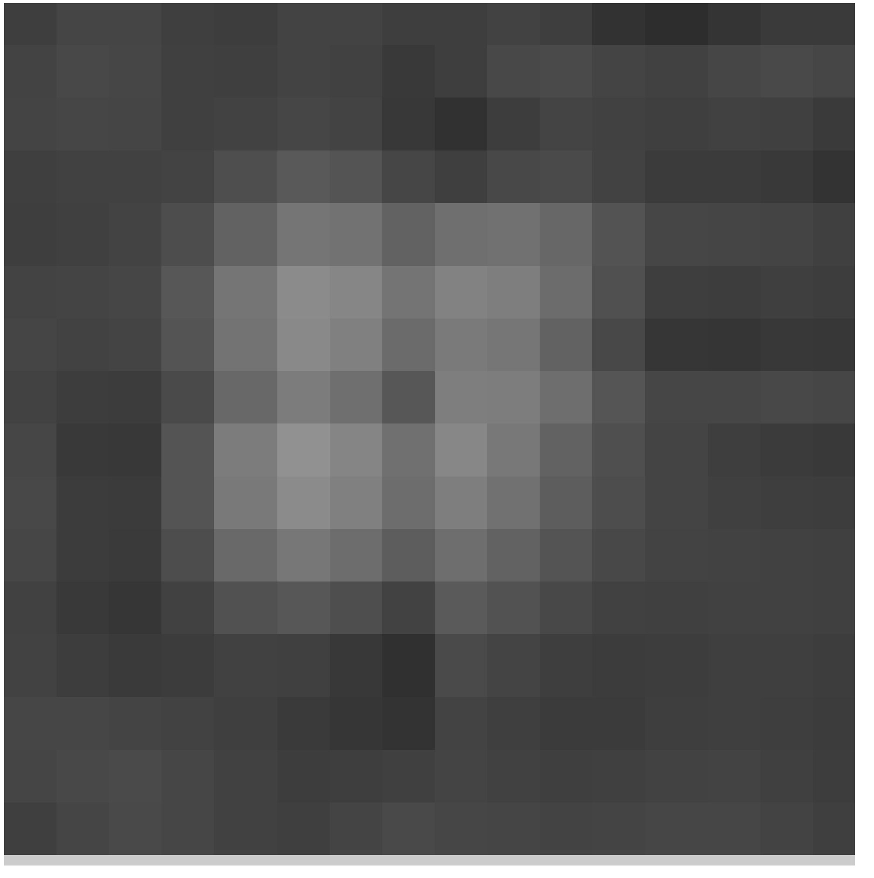
-1	9	0	-1	-3	8	1	12	-1	15	-13	-4	-9	-17	-1	2
5	6	-8	12	-9	13	-4	-9	-3	-3	3	12	10	20	8	-10
6	4	10	11	-14	-8	-12	-13	-16	-1	2	-13	-15	11	1	-2
-9	-13	-4	-10	8	19	28	13	11	9	19	-2	-15	-3	-9	-12
9	6	1	26	-24	-12	6	-16	-16	11	-18	12	7	-5	7	10
6	9	-6	11	-37	3	-4	-8	2	-14	-1	4	4	6	-7	-5
-4	-2	0	20	-3	8	29	7	-11	29	-5	7	-15	3	-21	3
12	-16	-8	17	-16	30	-32	-2	-1	11	-37	25	-10	-6	-4	8
0	-15	-9	4	33	-7	-17	9	47	-36	5	17	-1	5	-7	5
15	4	-20	17	-7	-20	-7	26	25	-26	-14	24	11	2	-17	-6
-2	7	-2	21	-30	-4	-2	-17	-37	-10	-10	6	-7	-5	-3	14
7	-15	-2	-12	21	27	9	10	20	20	9	-15	14	-12	15	-5
2	-12	5	-12	2	-5	-13	-6	6	16	-7	0	-13	-8	8	-1
-5	12	-2	-10	-2	-2	4	0	5	-14	1	-1	6	9	-2	0
-6	-4	6	16	11	-8	-12	6	-11	2	0	-10	10	0	-5	3
-1	0	-2	5	-15	-3	-9	0	-4	10	10	14	-9	0	13	-2

Comparison of Reconstruction Image

Original image



Compression image

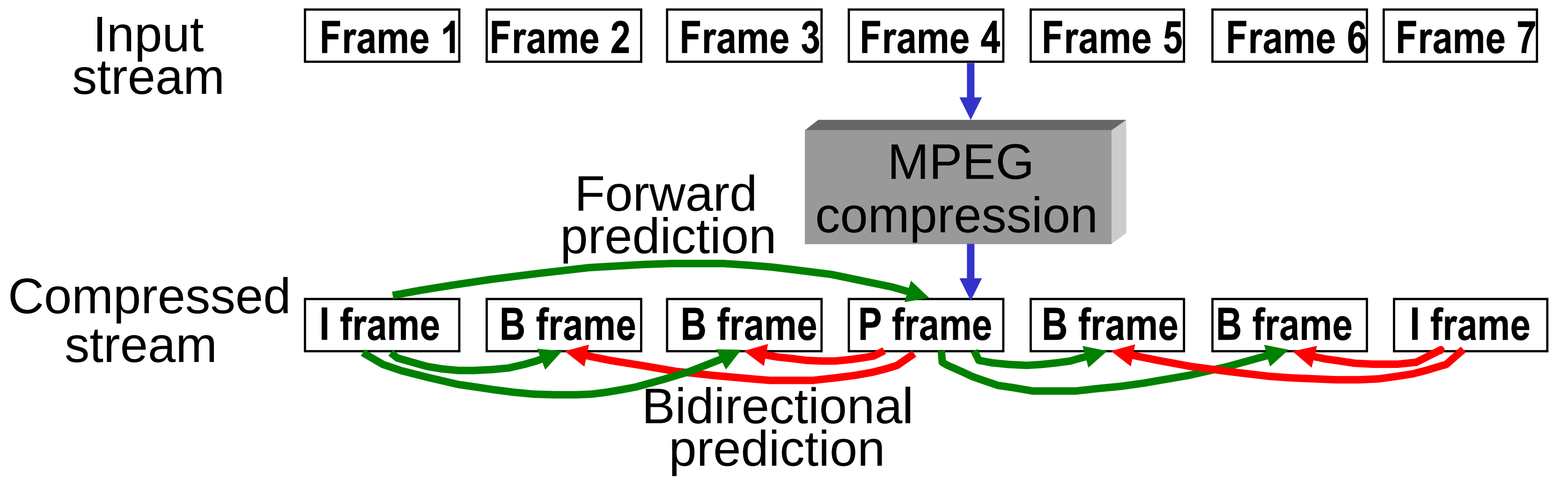


Video Compression (MPEG)

- **MPEG (Moving Picture Experts Group):** a moving picture can be simply approximated as a succession of **still images (frames)** displayed at some video rate
- Each of these frames can be compressed using the same **DCT-based** technique used in JPEG
  - Stopping at this point would be a **mistake**
- Two successive frames of video will contain **plenty of identical information**
  - It is unnecessary to send the same information twice
  - Should **remove** the **inter-frame redundancy** present in a video sequence
- MPEG takes this inter-frame redundancy into consideration

Video Compression (MPEG)

- MPEG takes a sequence of video frames as input and compresses them into three types of frames
  - **I frames (intrapicture)**
  - **P frames (predicted picture)**
  - **B frames (bidirectional predicted picture)**
- Each frame is compressed into one of these three frame types



Video Compression (MPEG)

- **I frames** can be thought of as **reference frames**
  - **Self-contained:** depending on neither earlier frames nor later frames
  - An I frame is simply the **JPEG compressed version** of the corresponding frame in the video source
- **P and B frames** are **not self-contained**
  - Specify **relative differences** from some reference frame
  - **P frame:** specifies the differences from the **previous** I frame
    - Depends on the preceding I frame

Video Compression (MPEG)

- **B frame:** gives an interpolation between the **previous** and **subsequent** I or P frames
  - Depends on both the preceding I or P frame and the subsequent I or P frame
- Because each B frame depends on a later frame in the sequence
  - The compressed frames are **not transmitted in sequential order**
  - The sequence **"I B B P B B I"** is transmitted as
    - **"I P B B I B B"**

Video Compression (MPEG)

- MPEG does not define the **ratio** of I frames to P and B frames
  - This ratio may vary depending on the **required compression** and **picture quality**
- Since MPEG coding is very expensive, it is normally done **offline** (not in real time)
  - For example, in a **video-on-demand system**, the video would be encoded and stored on disk ahead of time
- MPEG works I frames in units of **16×16 macroblocks**
- The P and B frames are also processed in units of **macroblocks**

Video Compression (MPEG)

- (Motion estimation) The information **captures the motion** of each macroblock
  - It shows in what **direction** and how **far** the macroblock moved relative to the reference frames
- If the motion picture is **changing too rapidly**
  - It makes sense **to give the intrapicture encoding** rather than a forward- or backward-predicted encoding

    ⇒ A **B frame** can use the same intracoding as is used in an I frame (no prediction is required)
  - Each macroblock in a **B frame** includes a **type field** that indicates which encoding is used for that macroblock
- MPEG typically achieves a compression ratio of **90-to-1**

Audio Compression (MP3)

- MPEG also defines a standard for compressing audio
- **CD-quality audio** is for high-quality audio
  - **Sampling rate: 44.1 KHz** (23 μs per sample)
  - Each sample is encoded by **16 bits**
  - For a stereo **(2-channel)** audio stream, the rate is **1.41Mbps**
- For **telephone-quality voice**: it has an 8 KHz sampling rate with 8-bit per sample ⇒ **64 Kbps**
- Some amount of compression is going to be required to transmit CD-quality audio over a **128-Kbps ISDN** line pair
- It is assumed that **49 bits** are used to encode each 16-bit sample
  - Including **synchronization** and **error correction** overhead
  - Actual bit rate is $49/16 \times 1.41$ Mbps = **4.32 Mbps**

Audio Compression (MP3)

- MPEG defines three layers of compression
  - **Layer III** is widely known as **MP3**
- MP3 uses techniques that are similar to those used by MPEG to compress video
  - Splits the audio stream into several **frequency subbands**
  - Each subband is broken into a sequence of blocks
  - Each block is **transformed** using a **modified DCT algorithm**, **quantized**, and **Huffman encoded**

Coding	Bit Rates	Compression Factor
Layer I	384 Kbps	4
Layer II	192 Kbps	8
Layer III	128 Kbps	12