



# **Digital Image Processing**

**Lecture #12**

**Ming-Sui (Amy) Lee**

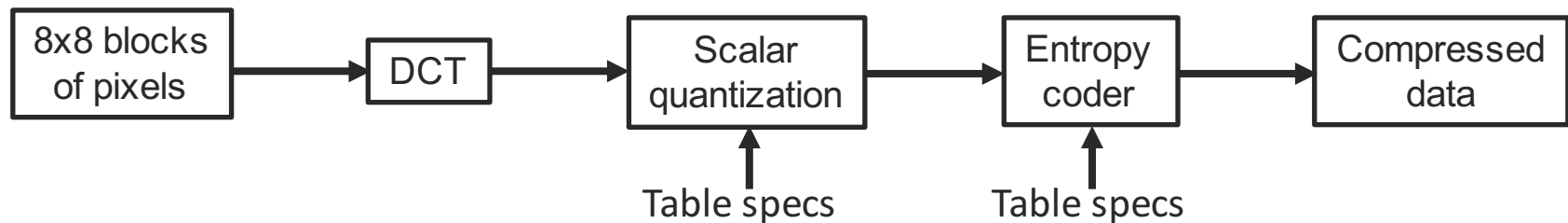


# Image Compression

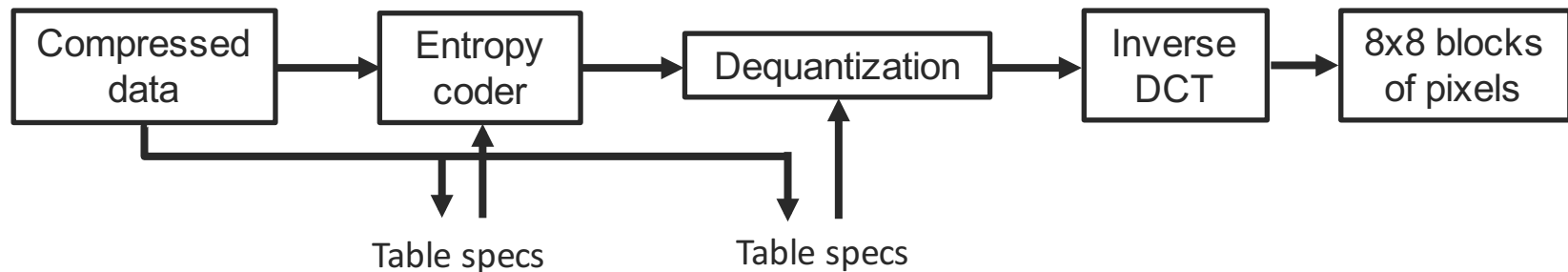
# Image Compression

## ■ JPEG Lossy Codec Scheme

**//Encoder//**



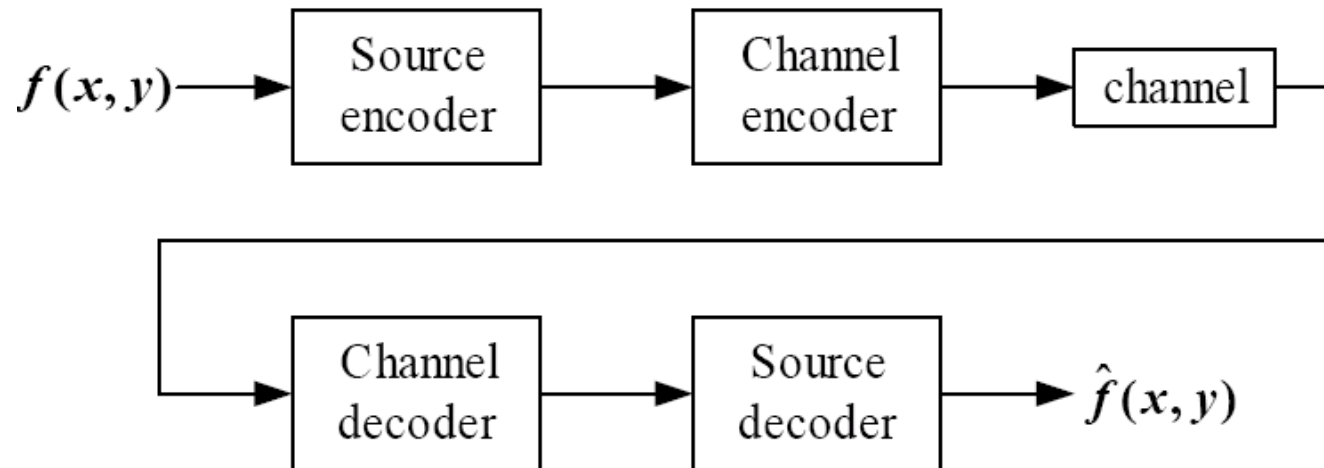
**//Decoder//**





# **Compression Model**

# Compression Model



- The encoder is composed of a source encoder and a channel encoder
  - Source encoder  
remove input redundancies
  - Channel encoder  
increase the noise immunity of the source encoder's output

# [ Compression Model ]

- **Source encoder**
  - reduce or eliminat any coding, interpixel, or psychovisual redundancies in an input image
- **Channel encoder**
  - If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted
  - As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of the “controlled redundancy”

# [ Compression Model ]

- Examples of source coding and channel coding
  - Source coding
    - Huffman code
    - Arithmetic code
  - Channel coding
    - Hamming code



# **Source Coding - Huffman Code**



# [Entropy]

- From information theory, the average number of bits needed to encode the symbols in a source  $S$  is always bounded by the entropy of  $S$

$$H = \sum p_i \cdot \log_2 \frac{1}{p_i}$$

- E.g.  $i = 2, p_1 = 1, p_2 = 0 \Rightarrow H = 0$   
 $p_1 = 1/2, p_2 = 1/2 \Rightarrow H = 1 \text{ bits / symbol}$

$$i = 4, p_1 = p_2 = p_3 = p_4 = 1/4 \Rightarrow H = 2 \text{ bits / symbol}$$

# [ Source Coding ]

- **Huffman code**
  - **Given the statistical distribution of the gray levels**
  - **Generate a code that is as close as possible to the minimum bound, entropy**
  - **A variable length code**

# [ Huffman Code ]

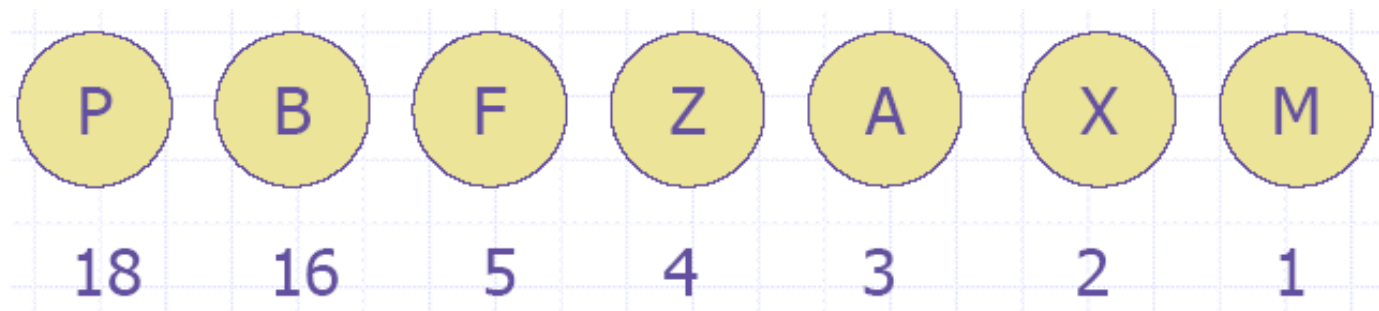
- **Five steps:**

- Find the gray-level probabilities for the image by finding the histogram
- Order the input probabilities (histogram magnitudes) from smallest to largest
- Combine the smallest two by addition  
(if multiple → Not a unique code)  
→ Repeat until one tree is formed
- For each node, assign 0 to the left, 1 to the right
- Traverse the tree from root to the leaf to generate the code

# [ Huffman Code ]

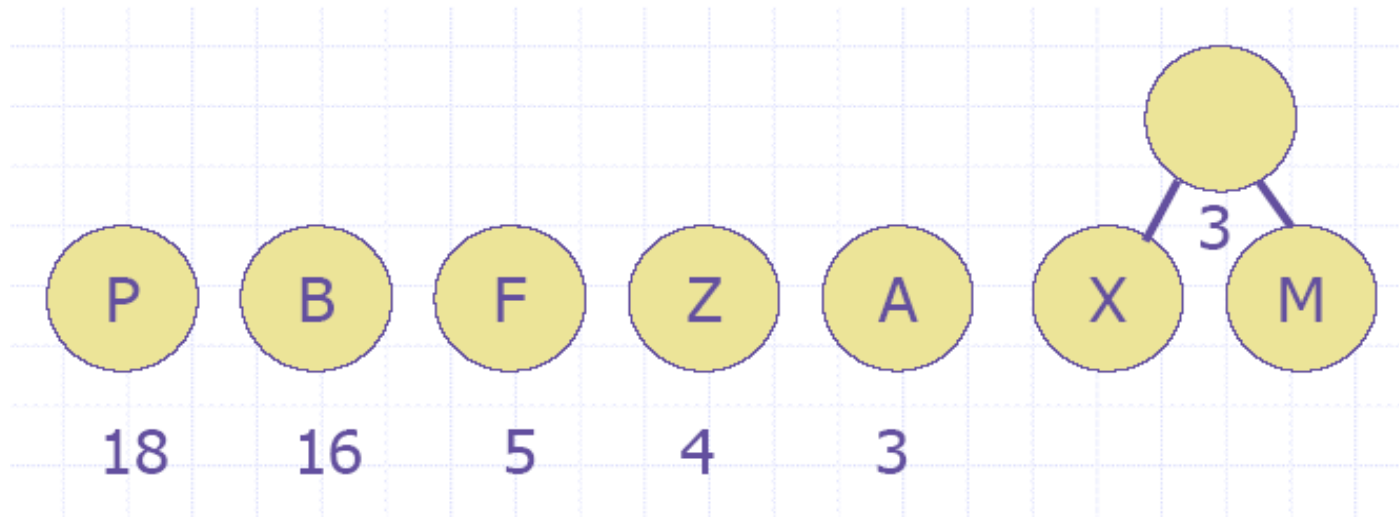
## □ Example

- ▣ Relative frequency of characters in a message text
- ▣ A larger number indicates higher frequency of use
  - ▣ Higher frequency → shorter code
  - ▣ Lower frequency → longer code



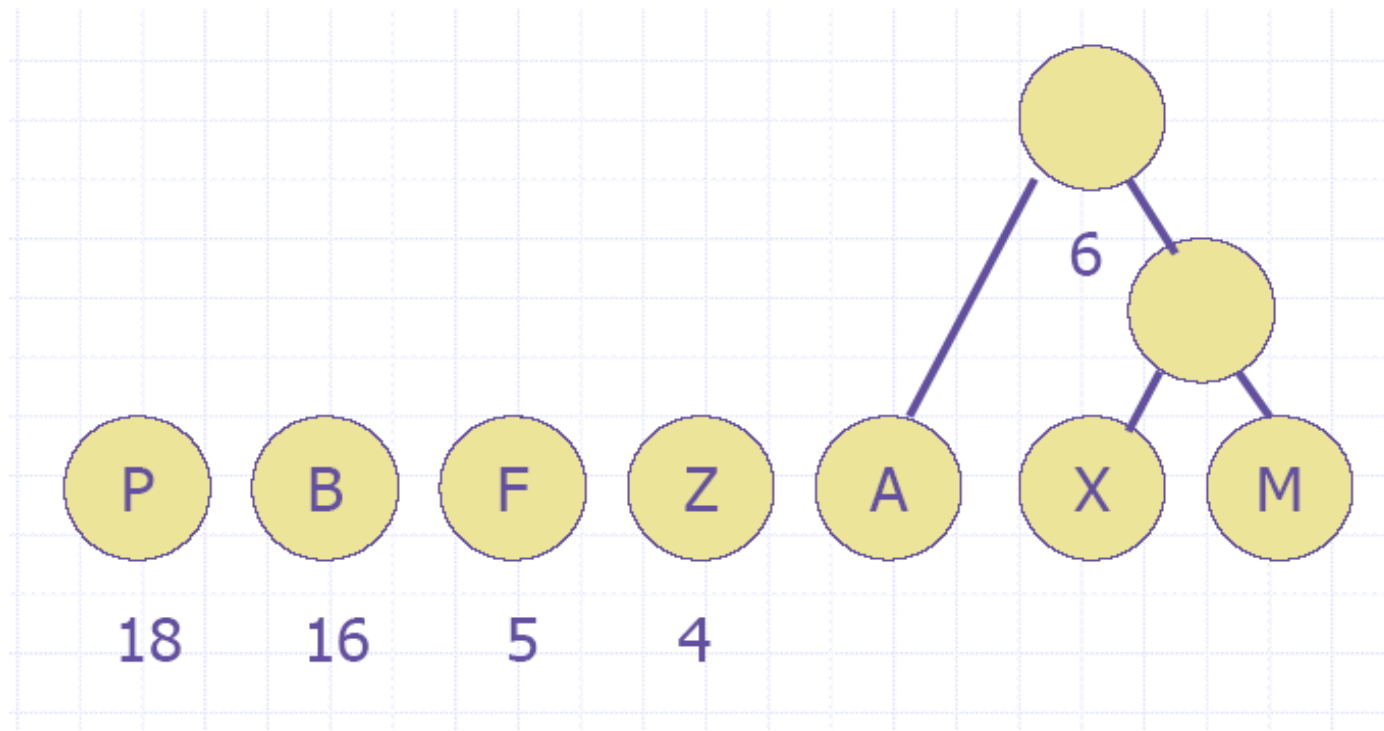
# [ Huffman Code ]

- Group lowest sum up the frequencies



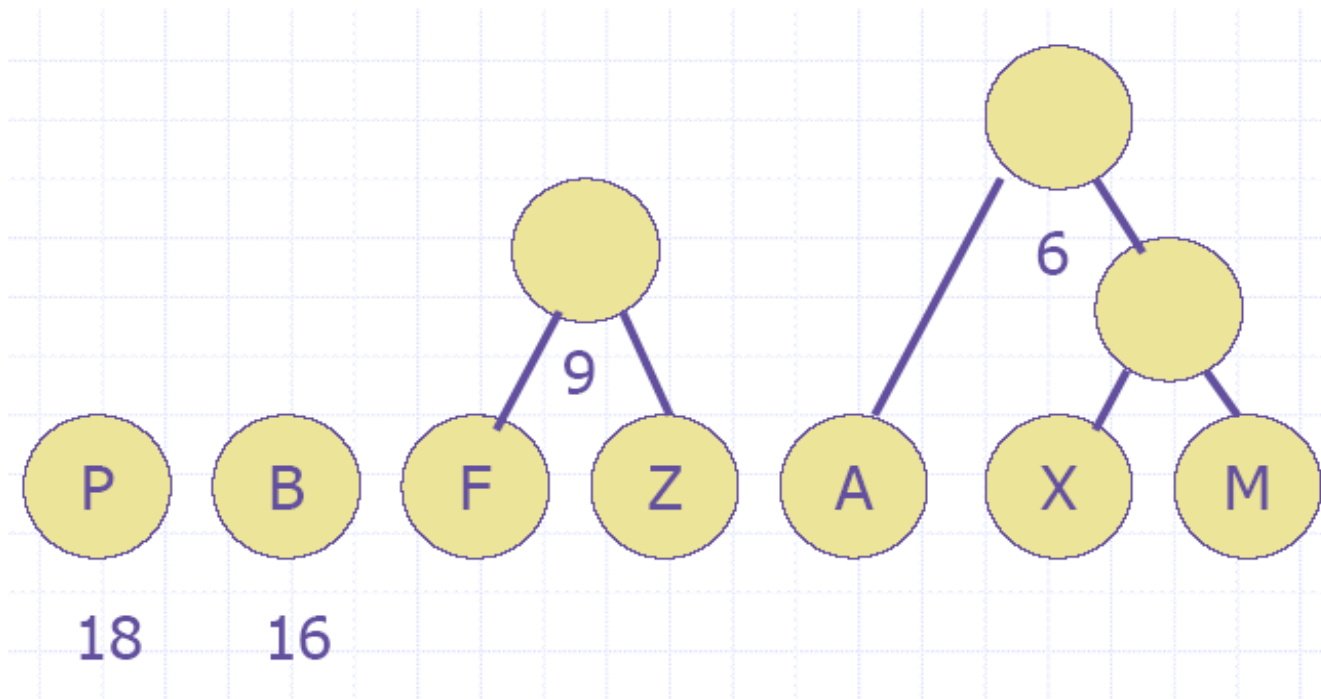
# Huffman Code

- Group next two lowest



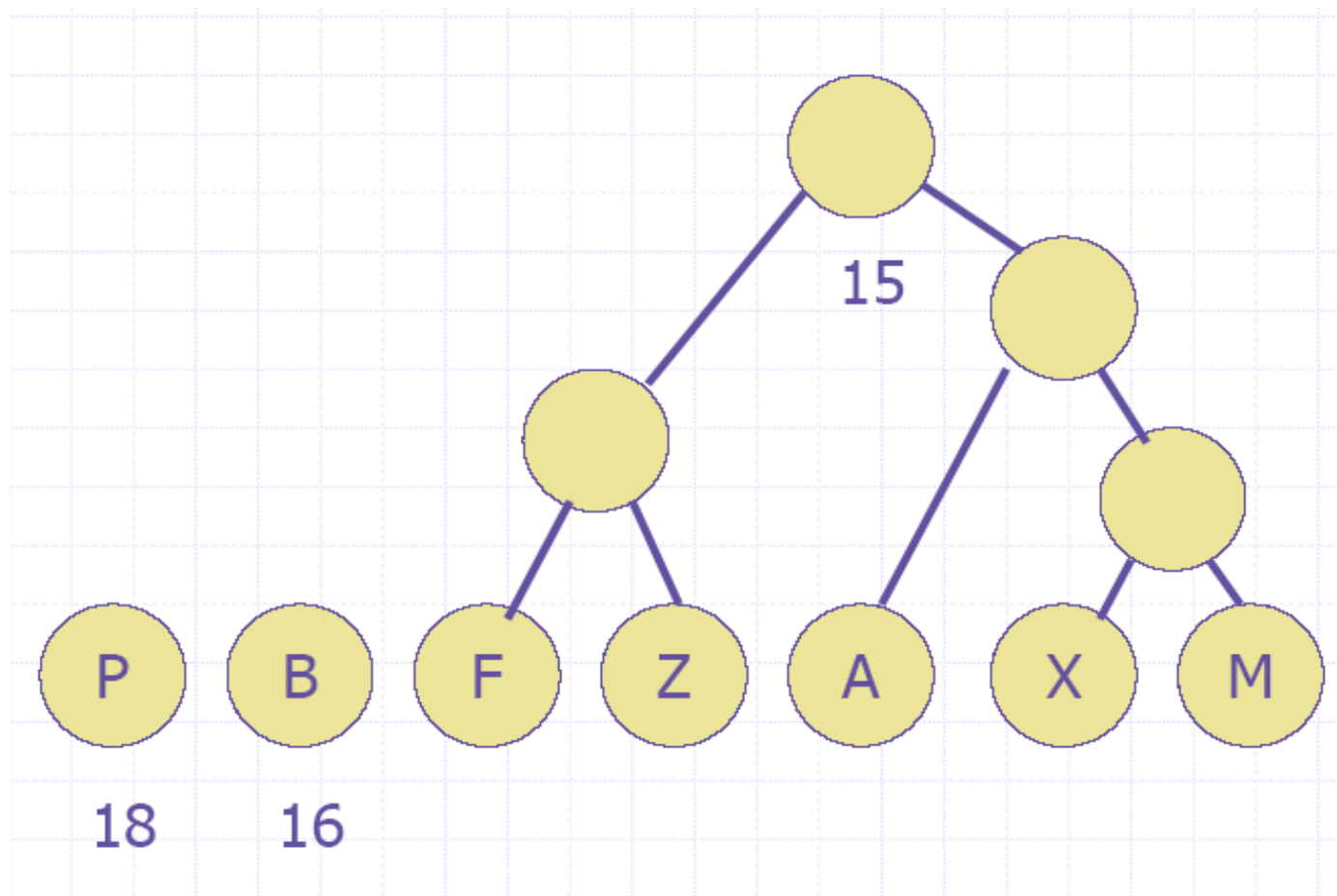
# Huffman Code

- Group next two lowest



# Huffman Code

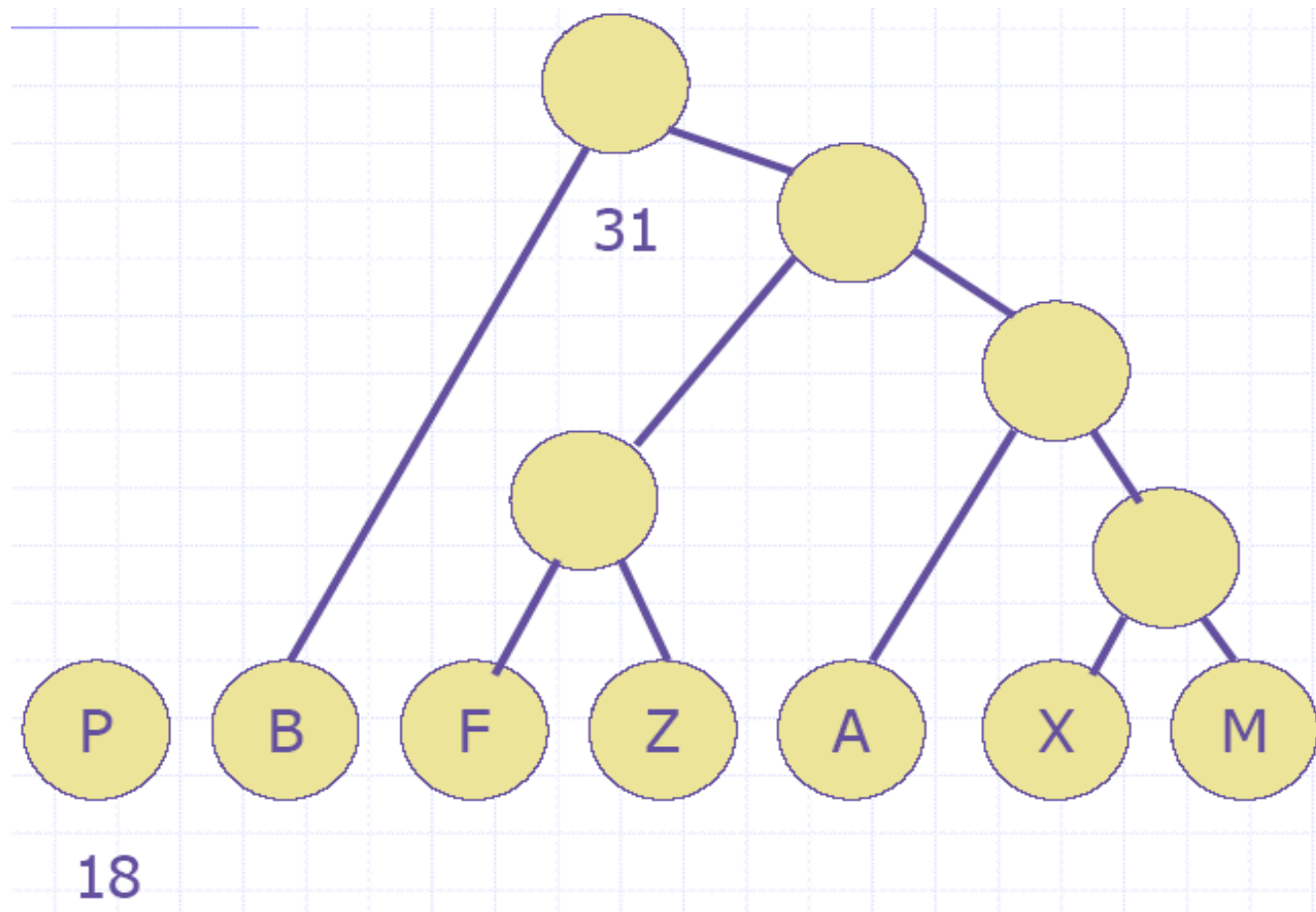
- Group next two lowest





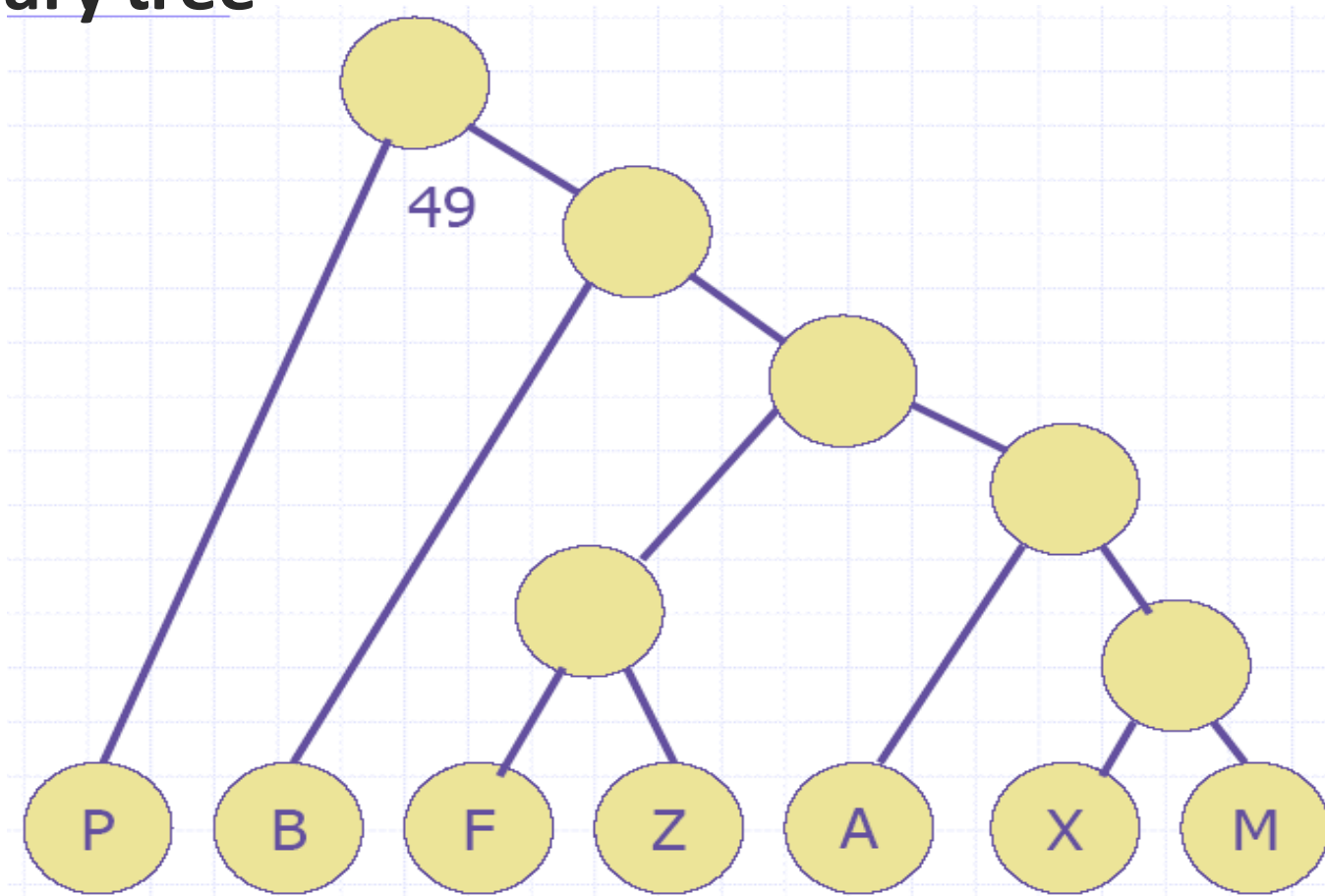
# Huffman Code

- Group next two lowest



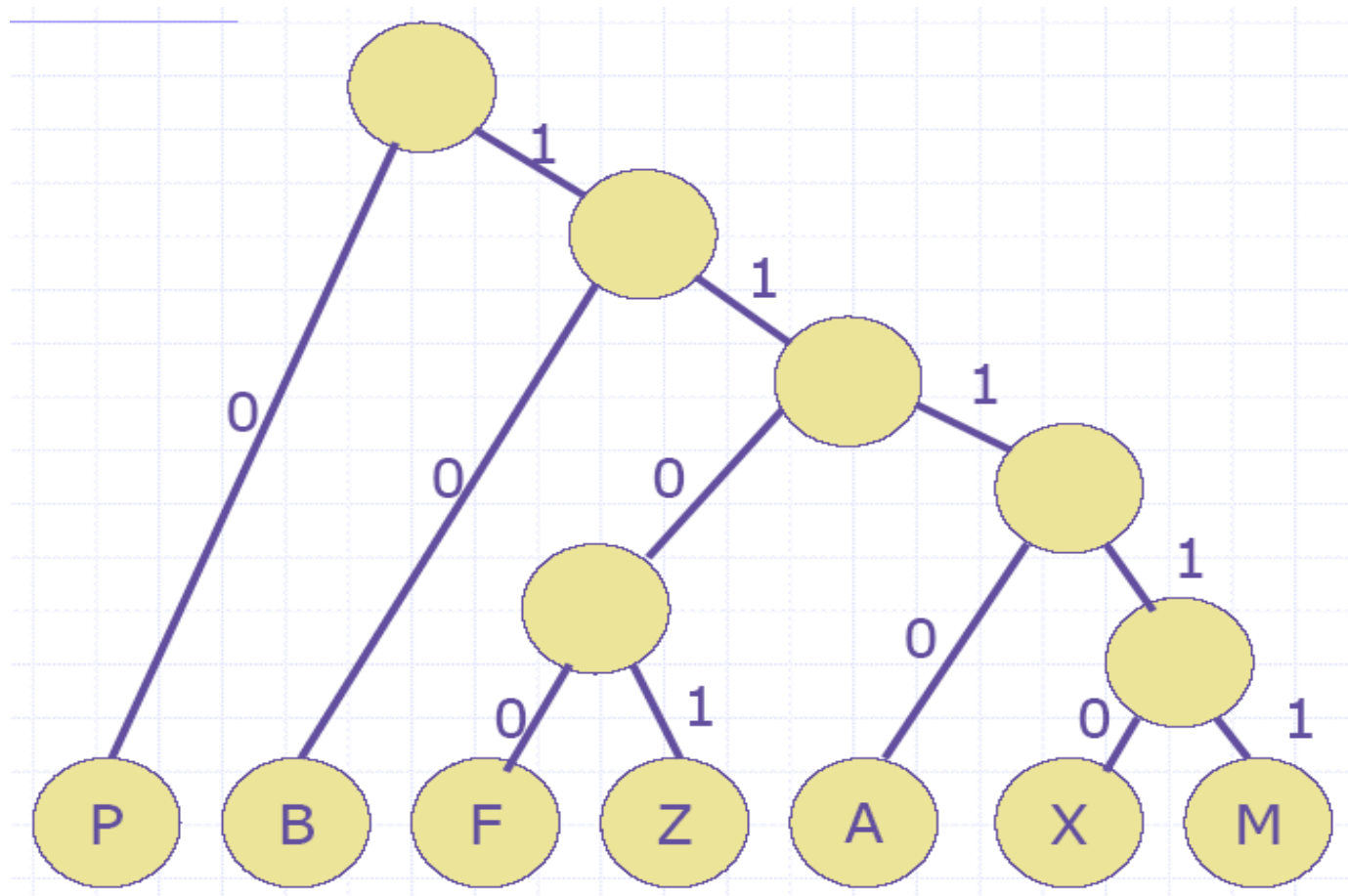
# Huffman Code

- Finally all characters are leaf nodes of a single binary tree



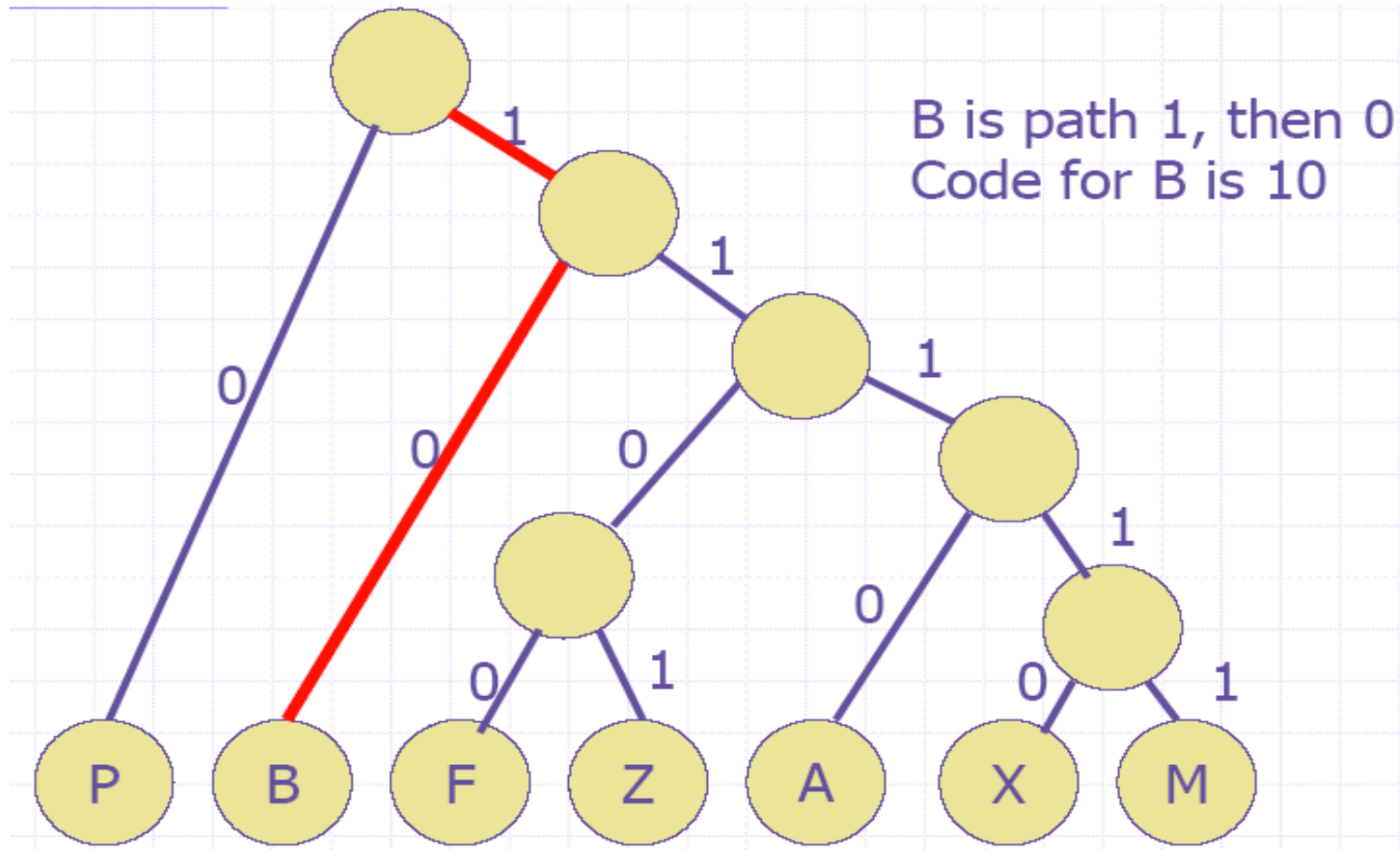
# Huffman Code

- Assign 0 to left and 1 to right



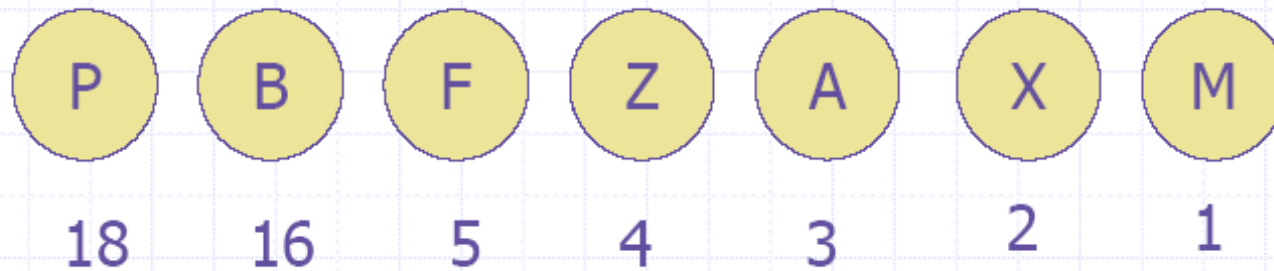
# Huffman Code

- Code for a character is the path to that character node from the root

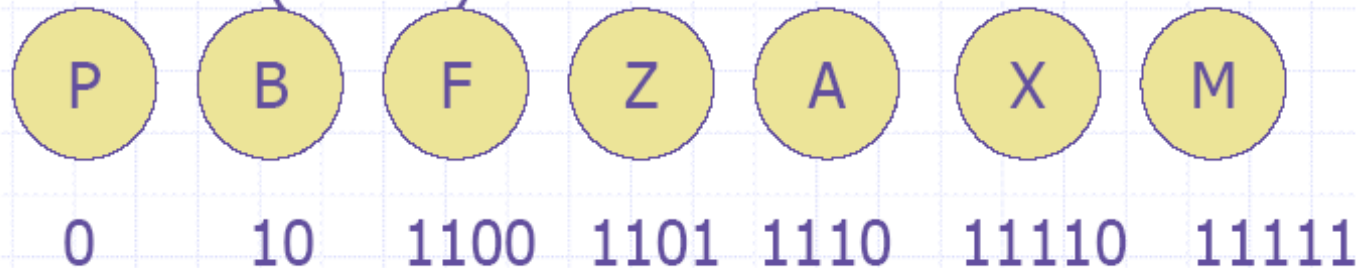


# Huffman Code

- Code for a character is the path to that character node from the root



Relative frequency of characters (above) have code shown (below).



Note that the most used character has the shortest code.

# Huffman Code

- Encode the following message

Message text is: PAMBAZ

Message coded is: 01110111111011101101

---

---

P	B	F	Z	A	X	M
0	10	1100	1101	1110	11110	11111

# Huffman Code

- How to decode the coded message?

Message coded is: 01110111111011101101

Read the code bit by bit, as the bits correspond to some Character, accept that series of bits as that character

---

---

P	B	F	Z	A	X	M
0	10	1100	1101	1110	11110	11111

# Huffman Code

- How to decode the coded message?

Message coded is: 01110111111011101101

0 11101111111011101101

P 11101111111011101101

P A 111111011101101

P A M 1011101101

P

B

F

Z

A

X

M

0

10

1100

1101

1110

11110

11111



# Huffman Code

- How to decode the coded message?

Message coded is: 01110111111011101101

P A M 1011101101

P A M B 11101101

P A M B A 1101

P A M B A Z

P

B

F

Z

A

X

M

0

10

1100

1101

1110

11110

11111

# Huffman Code

- Another example

- A character's code is found by starting at the root and following the branches that lead to that character. The code itself is the bit value of each branch on the path, taken in sequence

Frequency of characters

A	B	C	D	E
17	12	12	27	32

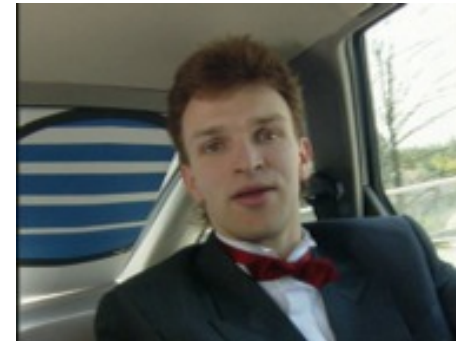
# [ Huffman Code ]

---

- Shorter codes are assigned to higher probability data items
- No code contains any other codes as a prefix
- Reading from the left-hand bit, each code is uniquely decodable

# [ Huffman Code ]

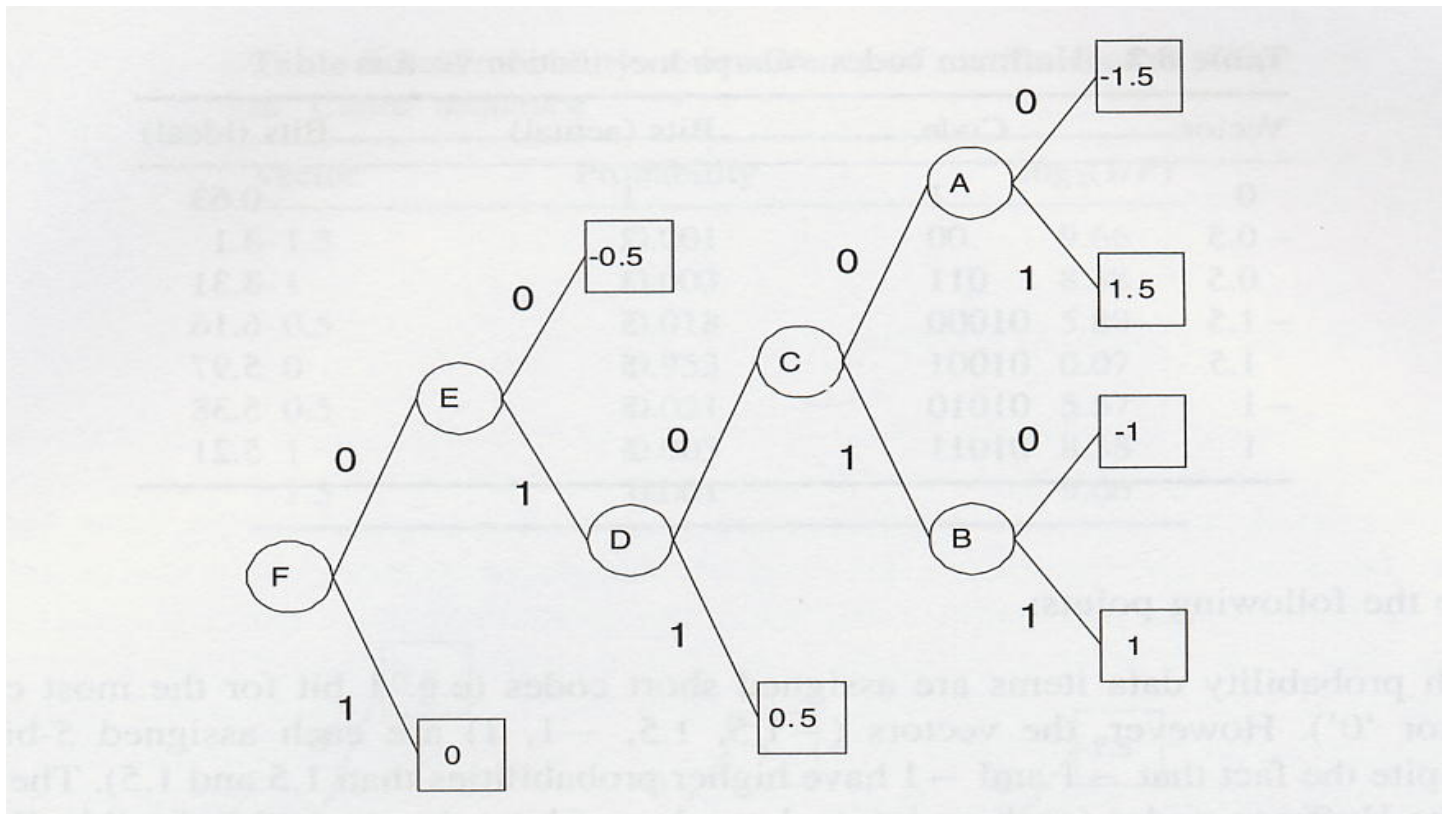
- Another example
  - Carphone sequence
    - Probability of motion vectors



Vector	Probability $P$	$\log_2(1/P)$
-1.5	0.014	6.16
-1	0.024	5.38
-0.5	0.117	3.10
0	0.646	0.63
0.5	0.101	3.31
1	0.027	5.21
1.5	0.016	5.97

# Huffman Code

- Another example
  - Huffman tree for Carphone sequence



# Huffman Code

- Another example
  - Huffman code for motion vectors of Carphone sequence

Vector	Code	Bits (actual)
0	1	1
-0.5	00	2
0.5	011	3
-1.5	01000	5
1.5	01001	5
-1	01010	5
1	01011	5

# Huffman Code

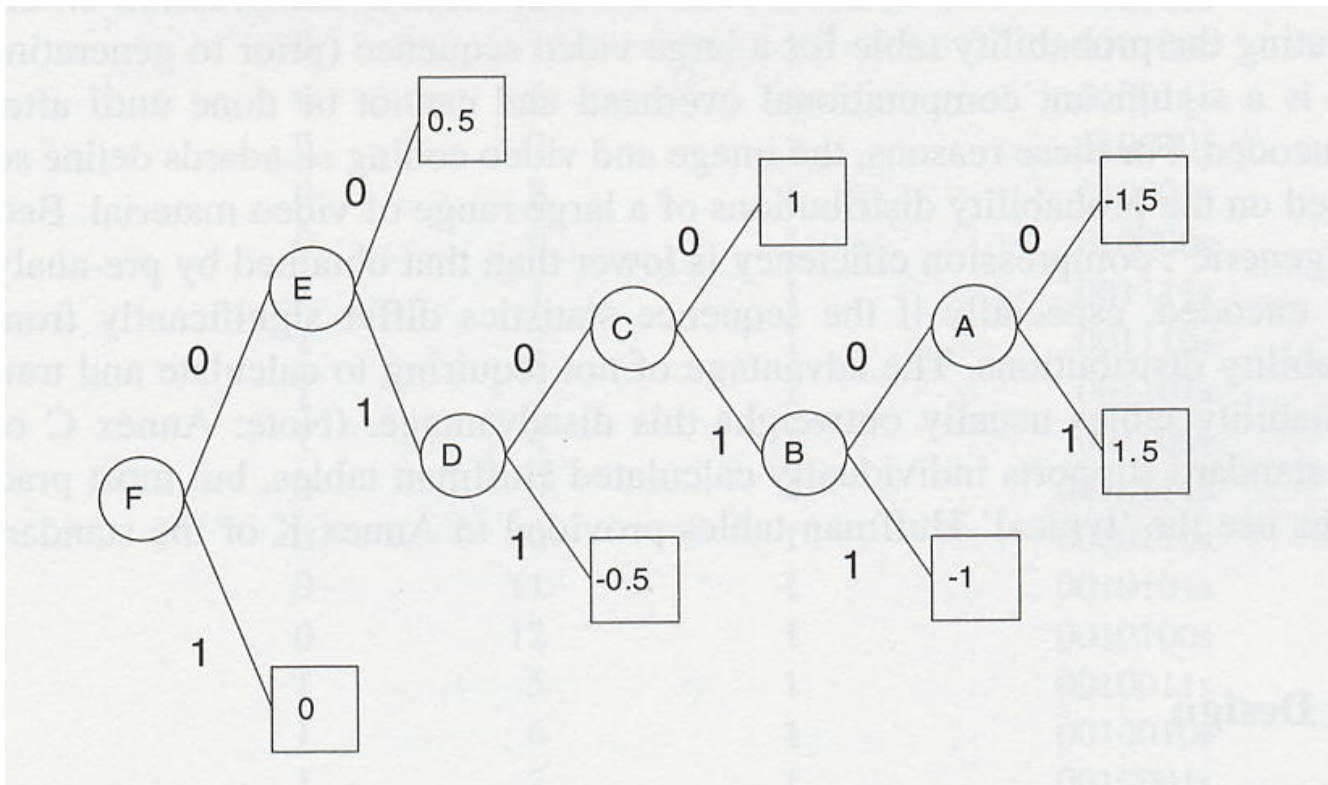
- Another example
  - Claire sequence
    - Probability of motion vectors



Vector	Probability	$\log_2(1/P)$
-1.5	0.001	9.66
-1	0.003	8.38
-0.5	0.018	5.80
0	0.953	0.07
0.5	0.021	5.57
1	0.003	8.38
1.5	0.001	9.66

# Huffman Code

- Another example
  - Huffman tree for Claire sequence





# [ Huffman Code ]

- Another example

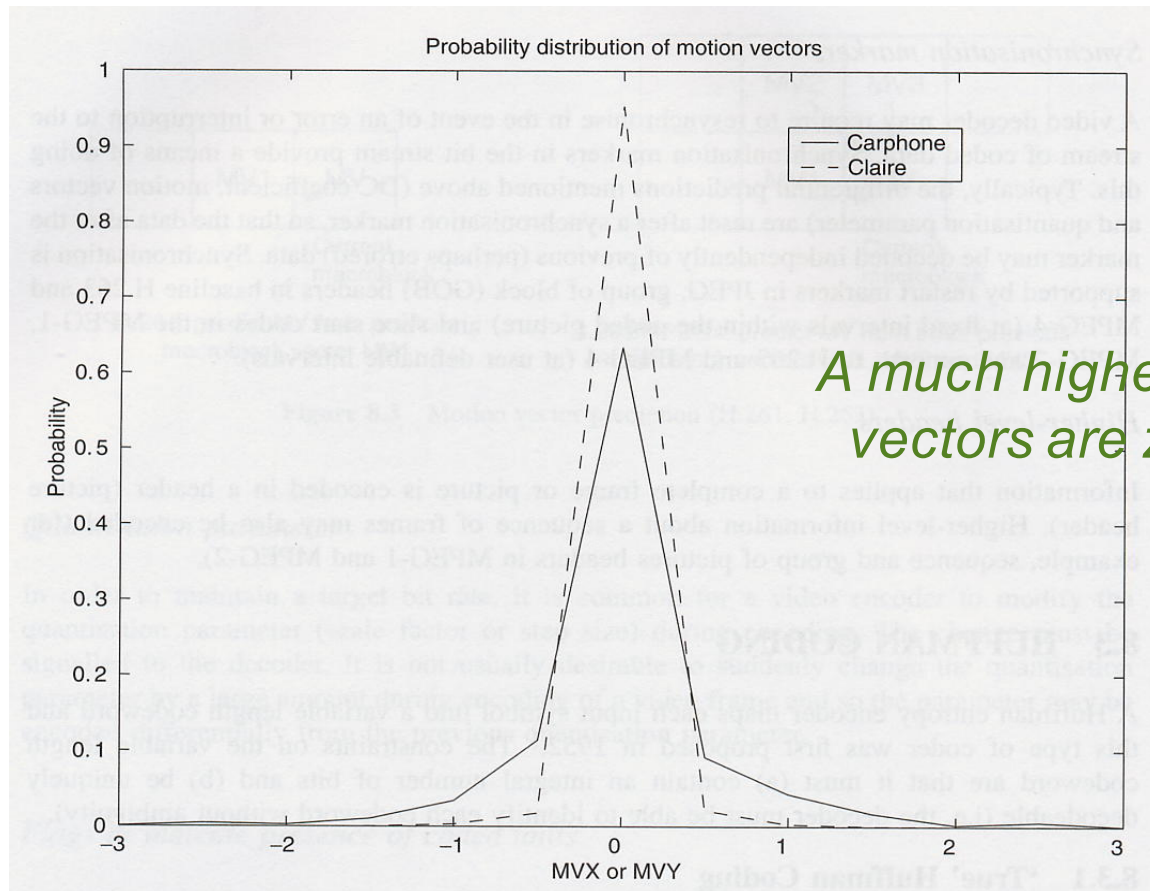
- Huffman code for motion vectors in Claire sequence

Vector	Code	Bits (actual)
0	1	1
0.5	00	2
-0.5	011	3
1	0100	4
-1	01011	5
-1.5	010100	6
1.5	010101	6

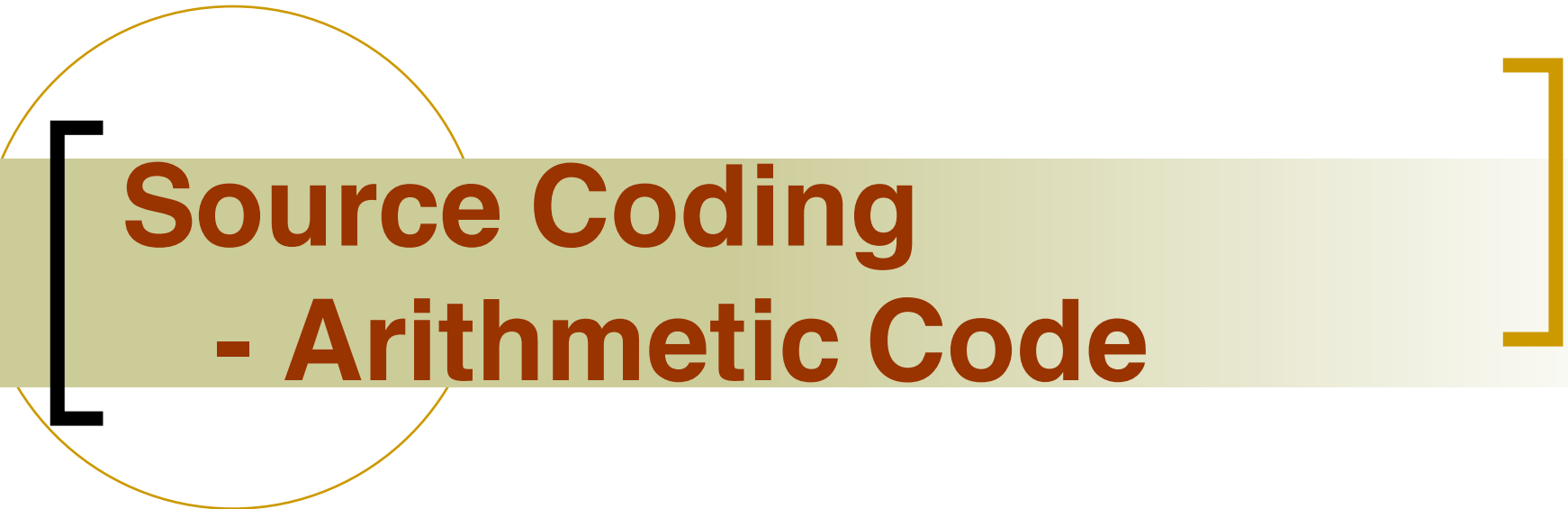
- To achieve optimum compression, a separate code table is required for each of the two sequences, *Carphone* and *Claire*

# Huffman Code

- Distribution of motion vectors
  - Carphone and Claire sequences



*A much higher proportion of vectors are zeros for Claire*



# **Source Coding - Arithmetic Code**

# Arithmetic Code

- Use the probability distribution of the data
- Convert sequence of data symbols into a single fractional/floating point number between 0 and 1
- The longer the sequence of symbols, the greater the precision required to represent the fractional number
- No direct correspondence between the code and the individual pixel values
- Theoretically achieves the maximum compression
- Quite different from Huffman coding, arithmetic coding is stream-based. It overcomes the drawbacks of Huffman coding

# Arithmetic Code

## ■ Arithmetic Coding Encoder

**BEGIN**

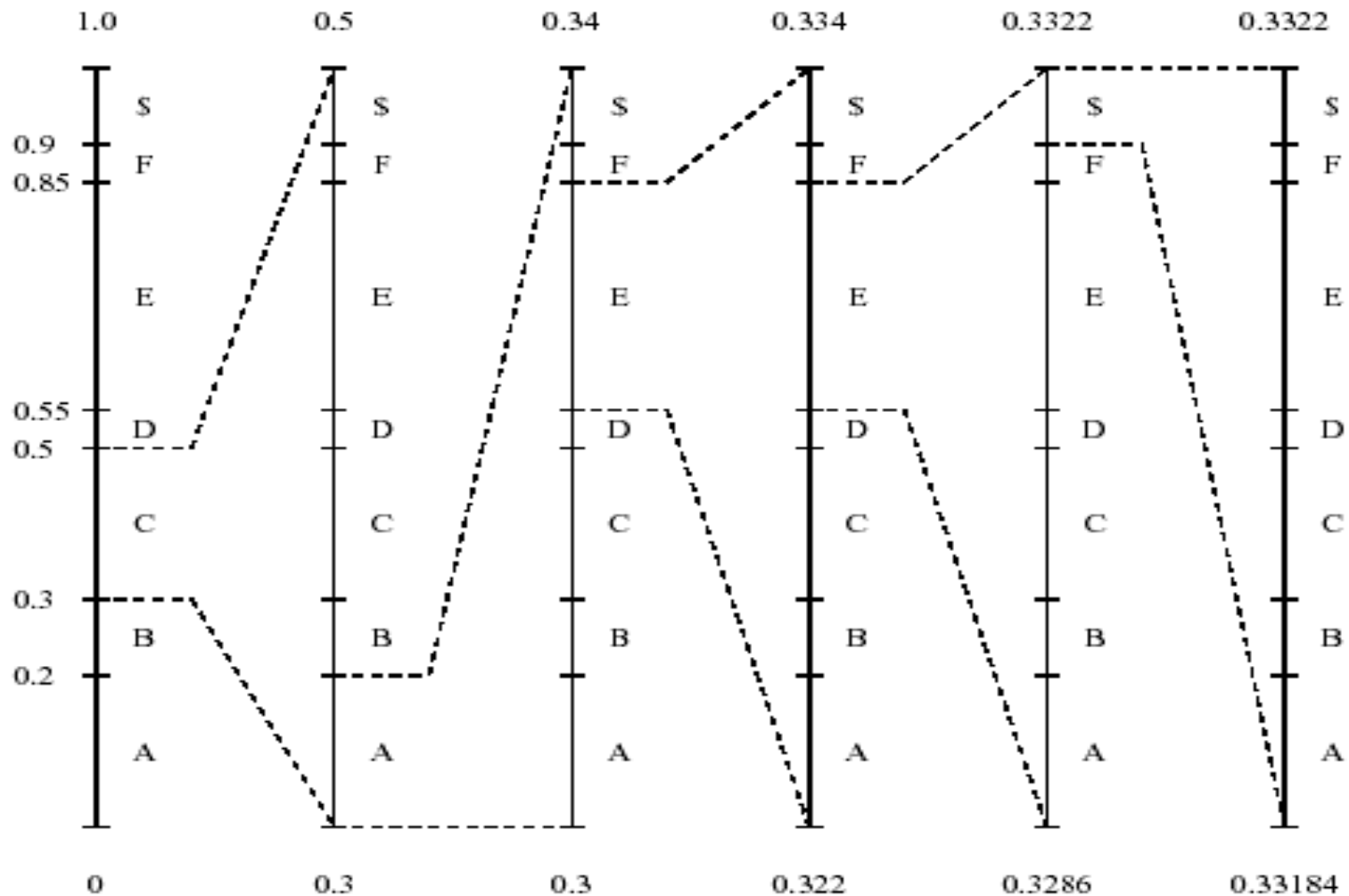
```
    low=0.0;      high=1.0;      range=1.0;
    while (symbol != terminator)
    {
        get (symbol);
        low=low+range*Range_low(symbol);
        high=low+range*Range_high(symbol);
        range=high-low;
    }
```

```
    output a code so that low<= code<high;
```

**END**

# Arithmetic Code

- Graphical display of shrinking ranges for “CAEE\$”



# Arithmetic Code

- Encode symbols “CAEE\$”
  - New low, high and range generated

Symbol	low	high	range
	0	1.0	1.0
C	0.3	0.5	0.2
A	0.30	0.34	0.04
E	0.322	0.334	0.012
E	0.3286	0.3322	0.0036
\$	0.33184	0.33220	0.00036

# Arithmetic Code

- **Generating codeword for encoder**

**BEGIN**

**code=0;**

**k=1;**

**while (value(code)<low)**

**{ assign 1 to the kth binary fraction bit**

**if (value(code) > high)**

**replace the kth bit by 0**

**k=k+1;**

**}**

**END**

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range [low, high). The above algorithm will ensure that the shortest binary codeword is found



# Arithmetic Code

## ■ Arithmetic coding decoder

**BEGIN**

get the binary code and convert to decimal value =  
value(code);

**Do**

{ find a symbol s so that

Range\_low(s)  $\leq$  value  $<$  Range\_high(s);

output s;

low=Range\_low(s);

high=Range\_high(s);

range =high-low;

value=[value-low]/range;

}

**Until** symbol s is a terminator

**END**

# Arithmetic Code

## ■ Decode symbols “CAEE\$”

```
...  
low=Range_low(s);  
high=Range_high(s);  
range =high-low;  
value=[value-low]/range;  
...
```

value	Output Symbol	low	high	range
0.33203125	C	0.3	0.5	0.2
0.16015625	A	0.0	0.2	0.2
0.80078125	E	0.55	0.85	0.3
0.8359375	E	0.55	0.85	0.3
0.953125	\$	0.9	1.0	0.1

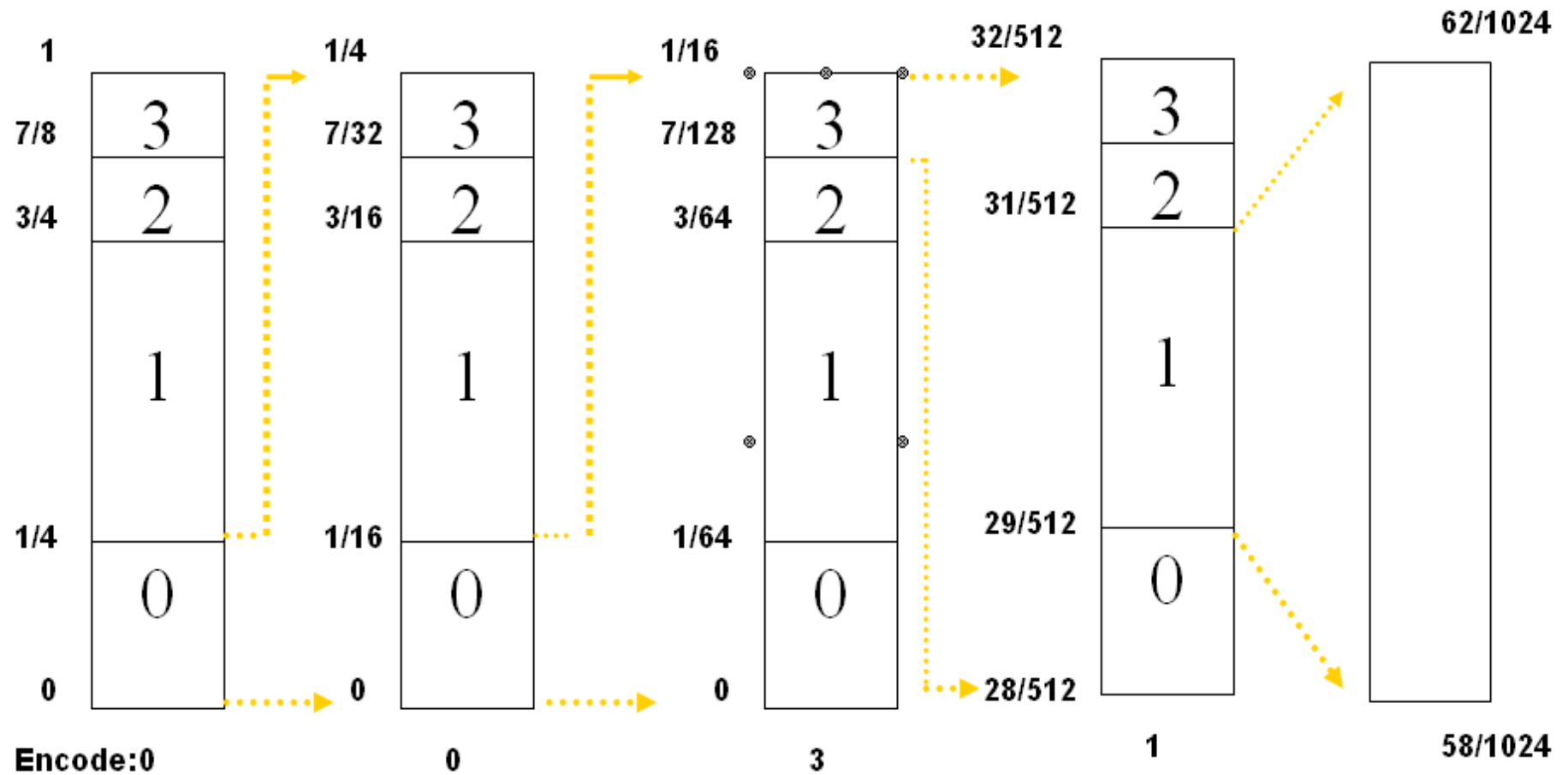
# Arithmetic Code

- Another example

**Probability Table**

<b>Pixel value</b>	<b>probability</b>	<b>Initial Subinterval</b>
<b>0</b>	<b><math>64/256=1/4</math></b>	<b><math>0 - 1/4</math></b>
<b>1</b>	<b><math>128/256=1/2</math></b>	<b><math>1/4 - 3/4</math></b>
<b>2</b>	<b><math>32/256=1/8</math></b>	<b><math>3/4 - 7/8</math></b>
<b>3</b>	<b><math>32/256=1/8</math></b>	<b><math>7/8 - 1</math></b>

# [ Arithmetic Code ]



Coding process for 0,0,3,1

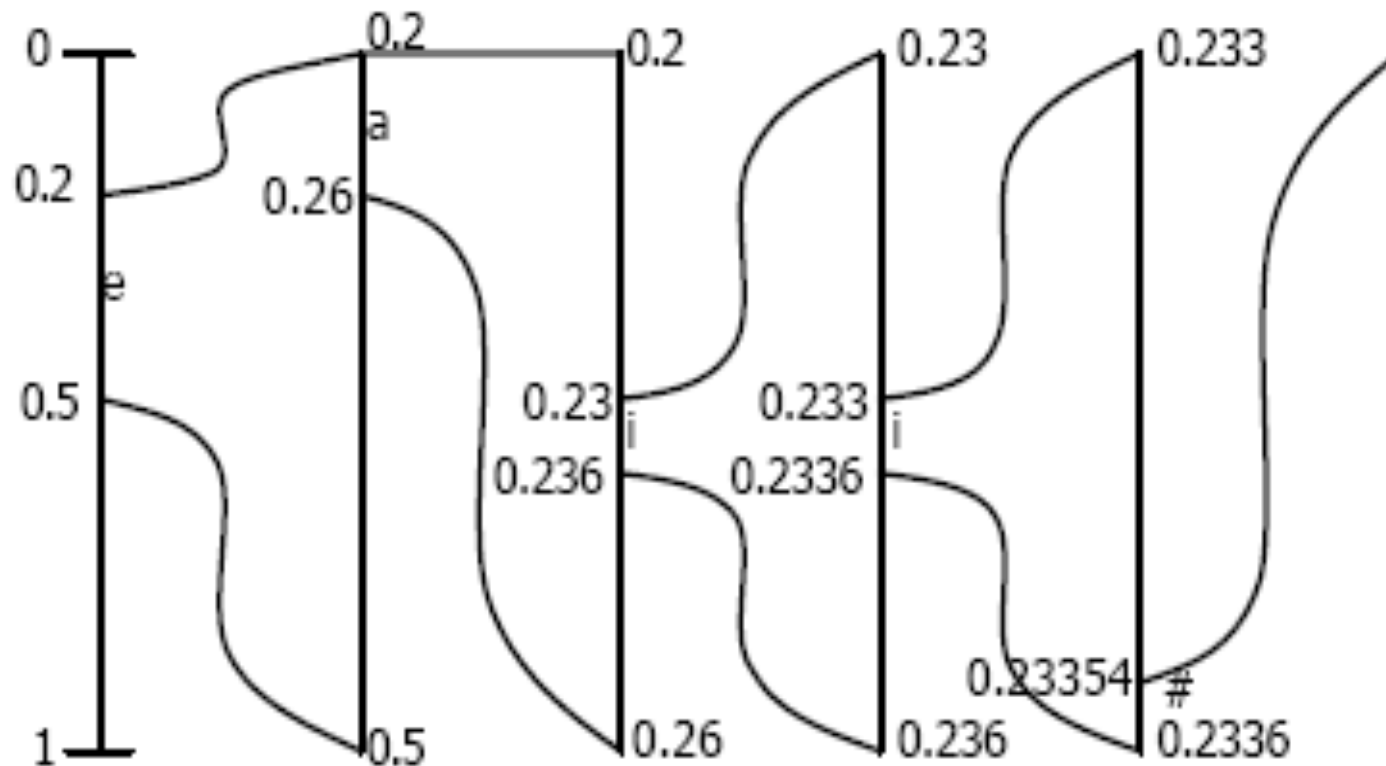
# [ Arithmetic Code ]

## ■ Example

Letter	Probability	Interval
a	0.2	$[0,0.2)$
e	0.3	$[0.2,0.5)$
i	0.1	$[0.5,0.6)$
o	0.2	$[0.6,0.8)$
u	0.1	$[0.8,0.9)$
#	0.1	$[0.9,1)$

# Arithmetic Code

- Message can be any  $x \in [0.23354, 0.2336)$

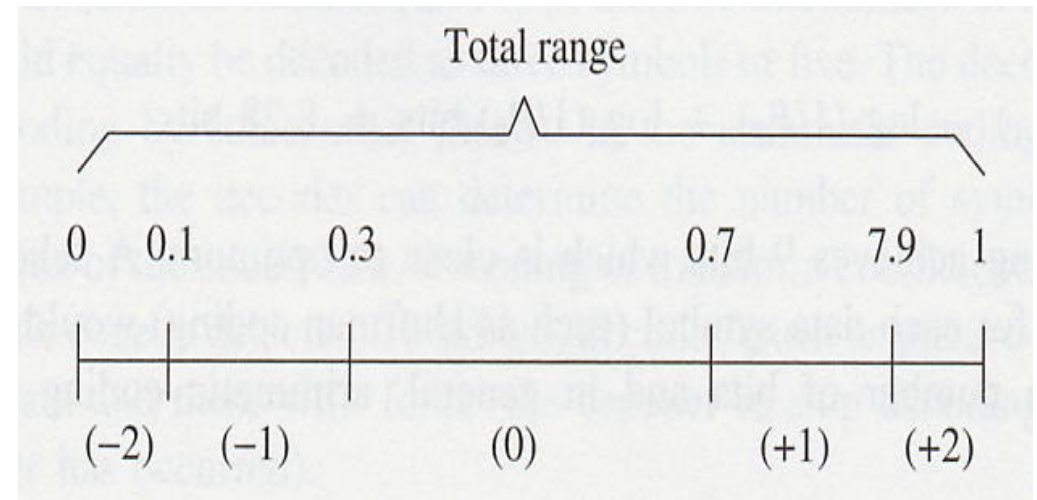


# Arithmetic Code

## ■ Example

- Each vector is assigned a subrange within the range 0-1 depending on its probability of occurrence

Vector	Probability	Subrange
-2	0.1	0-0.1
-1	0.2	0.1-0.3
0	0.4	0.3-0.7
1	0.2	0.7-0.9
2	0.1	0.9-1.0



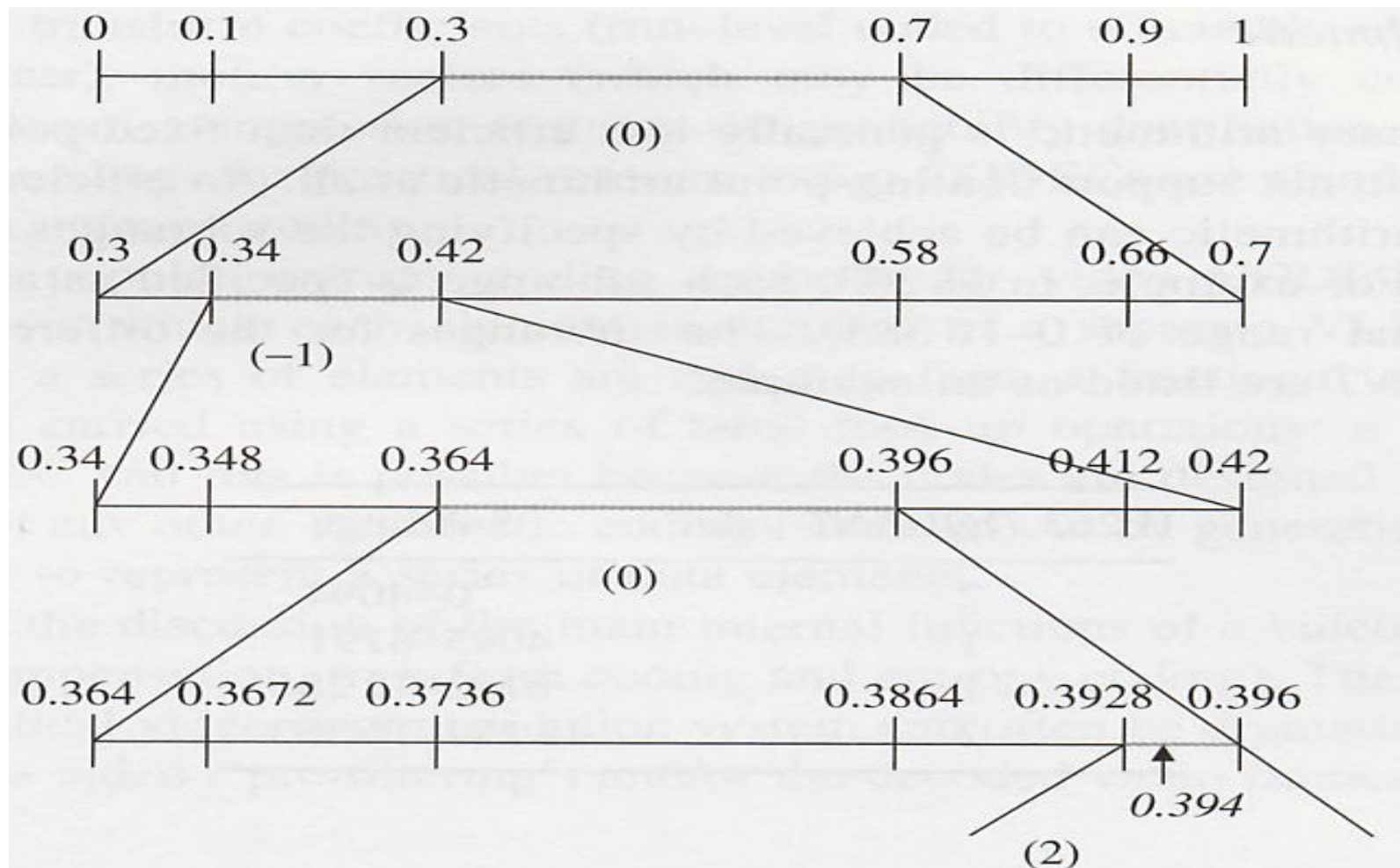
## Encoding procedure

Encoding procedure	Range (L $\rightarrow$ H)	Symbol	Subrange (L $\rightarrow$ H)	Notes
1. Set the initial range	0 $\rightarrow$ 1.0			
2. For the first data symbol, find the corresponding subrange (low to high).		(0)	0.3 $\rightarrow$ 0.7	
3. Set the new range (1) to this subrange	0.3 $\rightarrow$ 0.7			
4. For the next data symbol, find the subrange L to H		(-1)	0.1 $\rightarrow$ 0.3	This is the subrange within the interval 0-1
5. Set the new range (2) to this subrange within the previous range	0.34 $\rightarrow$ 0.42			0.34 is 10% of the range; 0.42 is 30% of the range
6. Find the next subrange		(0)	0.3 $\rightarrow$ 0.7	
7. Set the new range (3) within the previous range	0.364 $\rightarrow$ 0.396			0.364 is 30% of the range; 0.396 is 70% of the range
8. Find the next subrange		(2)	0.9 $\rightarrow$ 1.0	
9. Set the new range (4) within the previous range	0.3928 $\rightarrow$ 0.396			0.3928 is 90% of the range; 0.396 is 100% of the range



# Arithmetic Code

## ■ Example



## Decoding procedure

The sequence of subranges (and hence the sequence of data symbols) can be decoded from this number as follows.

Decoding procedure	Range	Subrange	Decoded symbol
1. Set the initial range	$0 \rightarrow 1$		
2. Find the subrange in which the received number falls. This indicates the first data symbol		$0.3 \rightarrow 0.7$	(0)
3. Set the new range (1) to this subrange	$0.3 \rightarrow 0.7$		
4. Find the subrange <i>of the new range</i> in which the received number falls. This indicates the second data symbol		$0.34 \rightarrow 0.42$	(-1)
5. Set the new range (2) to this subrange within the previous range	$0.34 \rightarrow 0.42$		
6. Find the subrange in which the received number falls and decode the third data symbol		$0.364 \rightarrow 0.396$	(0)
7. Set the new range (3) to this subrange within the previous range	$0.364 \rightarrow 0.396$		
8. Find the subrange in which the received number falls and decode the fourth data symbol		$0.3928 \rightarrow 0.396$	(2)

# Arithmetic Code

- An optional alternative to Huffman coding in several video coding standards(e.g. H.263, MPEG-4,H.26L)
- Pre-calculated subranges are defined by the standard (based on typical probability distributions)
  - Advantage: avoiding the need to calculate and transmit probability distributions
  - Disadvantage: suboptimal compression
- Syntax-based Arithmetic coding (SAC) in H.263 (about 5% decrease in bit-rate)

# Arithmetic Code

- **Patent Issues**

- IBM's Q-coder
- Q-Coder adaptive binary arithmetic coder
- Some developers of commercial video coding systems have avoided the use of arithmetic coding because of concerns about potential patent infringement
- H.264/AVC
  - Arithmetic coding is optional



# **Channel Coding - Hamming Code**



# [Channel Coding]

- **Hamming code**
  - Popular error correcting code in RAM
  - Richard Hamming 1950
- **In Hamming code**
  - $k$  bits parity code (check bit) and
  - $n$  data code (data bit)
  - form a  $(n+k)$  data word
  - Allocate parity codes to the positions of power of 2
  - May be applied to any length of code

# Channel Coding

## ■ Hamming code

- Example: 8 bits data 11000100

1	2	3	4	5	6	7	8	9	10	11	12
$P_1$	$P_2$	1	$P_4$	1	0	0	$P_8$	0	1	0	0

- $P_1$ =the XOR of bit(3, 5, 7, 9, 11) =  $1 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 0$
- $P_2$ =the XOR of bit(3, 6, 7, 10, 11) =  $1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 = 0$
- $P_4$ =the XOR of bit(5, 6, 7, 12) =  $1 \oplus 0 \oplus 0 \oplus 0 = 1$
- $P_8$ =the XOR of bit(9, 10, 11, 12) =  $0 \oplus 1 \oplus 0 \oplus 0 = 1$

The exclusive-OR operation perform an odd function

# Channel Coding

## ■ Hamming code

- Example: 12 bits data word

1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	1	0	0	1	0	1	0	0

- Check:

- $C1 = \text{the XOR of bits } (1,3,5,7,9,11)$
- $C2 = \text{the XOR of bits } (2,3,6,7,10,11)$
- $C4 = \text{the XOR of bits } (4,5,6,7,12)$
- $C8 = \text{the XOR of bits } (8,9,10,11,12)$
- $C = C8C4C2C1 = 0\ 0\ 0\ 0$  (no error occur)



# Channel Coding

## ■ Check (cont'd)

- If  $C \neq 0$ , then the 4-bit binary number formed by the check bits gives the position of the erroneous bit.

For example :

1	2	3	4	5	6	7	8	9	10	11	12	Bit position
0	0	1	1	1	0	0	1	0	1	0	0	No error
1	0	1	1	1	0	0	1	0	1	0	0	Error in bit 1
0	0	1	1	0	0	0	1	0	1	0	0	Error in bit 5

	$C_8$	$C_4$	$C_2$	$C_1$
For no error:	0	0	0	0
With error in bit 1:	0	0	0	1
With error in bit 5:	0	1	0	1

$C_1$  = the XOR of bits (1,3,5,7,9,11)     $C_4$  = the XOR of bits (4,5,6,7,12)  
 $C_2$  = the XOR of bits (2,3,6,7,10,11)     $C_8$  = the XOR of bits (8,9,10,11,12)

# [Channel Coding]

- If a nonzero value is found, the decoder simply complements the codeword bit position indicated by the parity word
- k bits v.s. n data bits

Number of check nits, k	Range of data bits, n
3	2-4
4	5-11
5	12-26
6	27-57
7	58-120

# Channel Coding

- Hamming code

- Linear code
- Can be computed in the form of matrices

- Example: 4 data bits: 1011

- Code generator matrix

$$G = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Parity-check matrix

$$C = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

# Channel Coding

## ■ Hamming code

### ○ Example (cont'd)

$$h = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 2 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Case1:

$$p = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Case2:

$$p = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

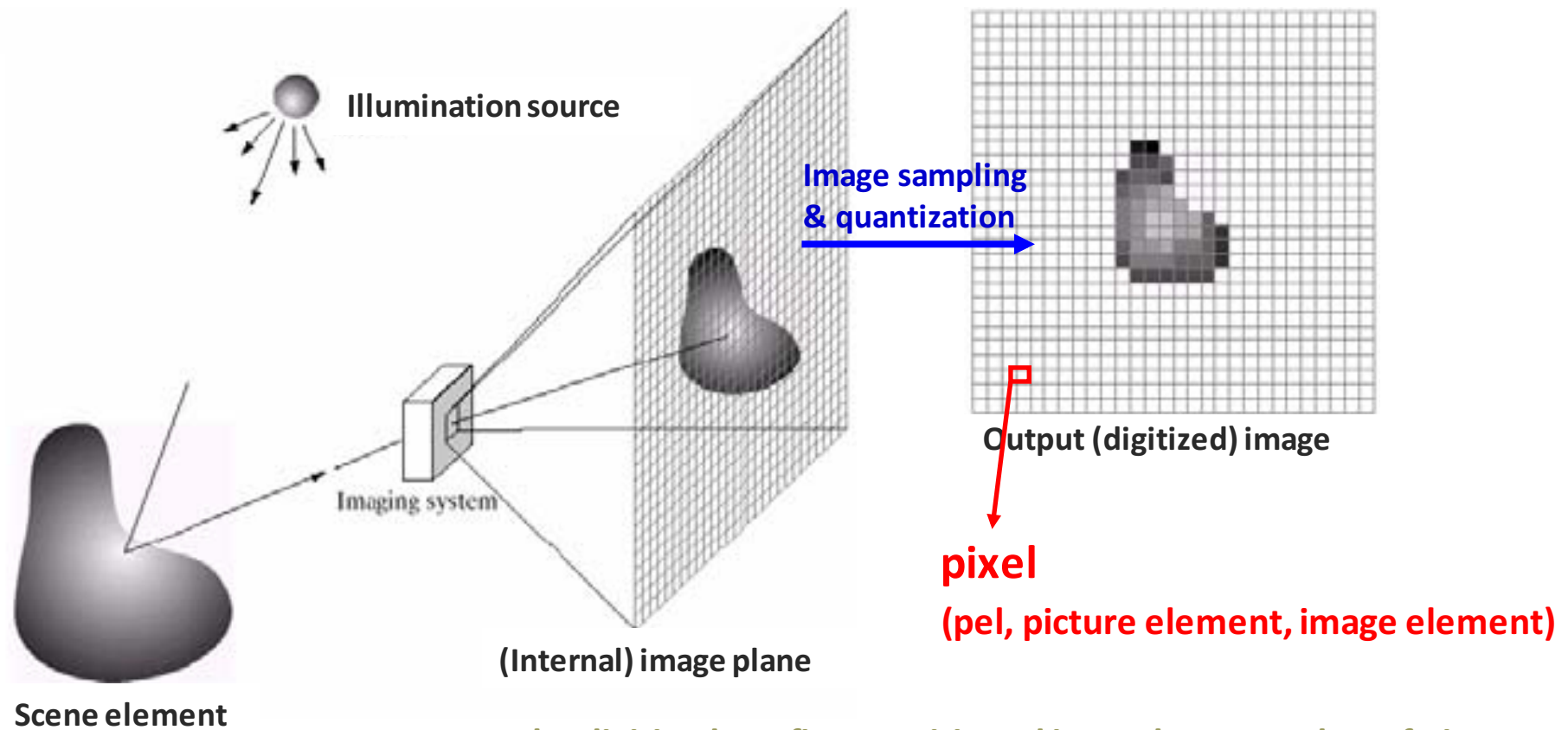


# **Image/Video Compression**

# [ Digital Images/Videos ]

- Good quality for storage and transmission interactivity
- Variable-rate transmission on demand
- Easy software conversion from one standard to another
- Integration of various video applications
- Editing capabilities, such as cutting and pasting, zooming, removal of noise and blur
- Robustness to channel noise and ease of encryption
- Trend of DSP VLSI (software to hardware solutions)

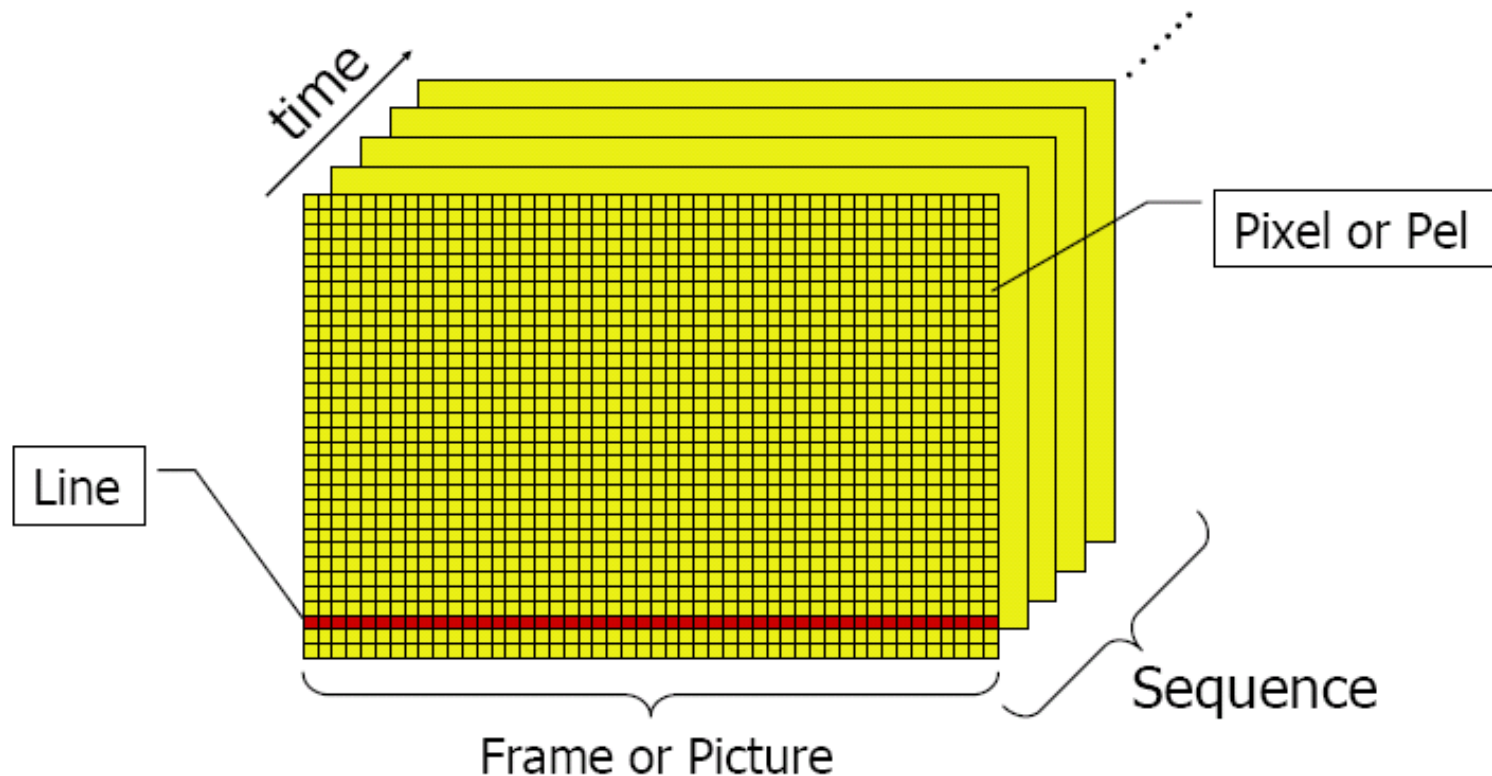
# [ An Image (A Frame) ]



Frames to be digitized are first partitioned into a large number of picture elements or pels for short, which are typically arranged in row and columns. A row of pels is called a scanline.

# Images and Videos

- Video: a sequence of images/frames





# Needs for Image/Video Compression

## ■ Without compression

- **Test pictures**

$720(\text{pels}) \times 576(\text{lines}) \times 1.5\text{bytes} = 4.977\text{Mbytes} = 77.8\text{s}$  at 64kbytes/s

- **Visual telephony** (e.g. CIF @15 frames/s)

$352(\text{pels}) \times 288(\text{lines}) \times 15(\text{frames/s}) \times 1.5\text{bytes} = 18.25\text{Mbytes/s}$

- **Digital TV** (CCIR601 4:2:0 @30 frames/s)

$720(\text{pels}) \times 480(\text{lines}) \times 30(\text{frames/s}) \times 1.5\text{bytes} = 124.4\text{Mbytes/s}$

- **HDTV**

$1280(\text{pels}) \times 720(\text{lines}) \times 60(\text{frames/s}) \times 1.5\text{bytes} = 663.6\text{Mbytes/s}$

- E.g. A 'television-quality' digital video signal: 216Mbytes/s

a 2-hour movie (uncompressed)

over 194 Gbytes → 42 DVDs or 304 CD-ROMs

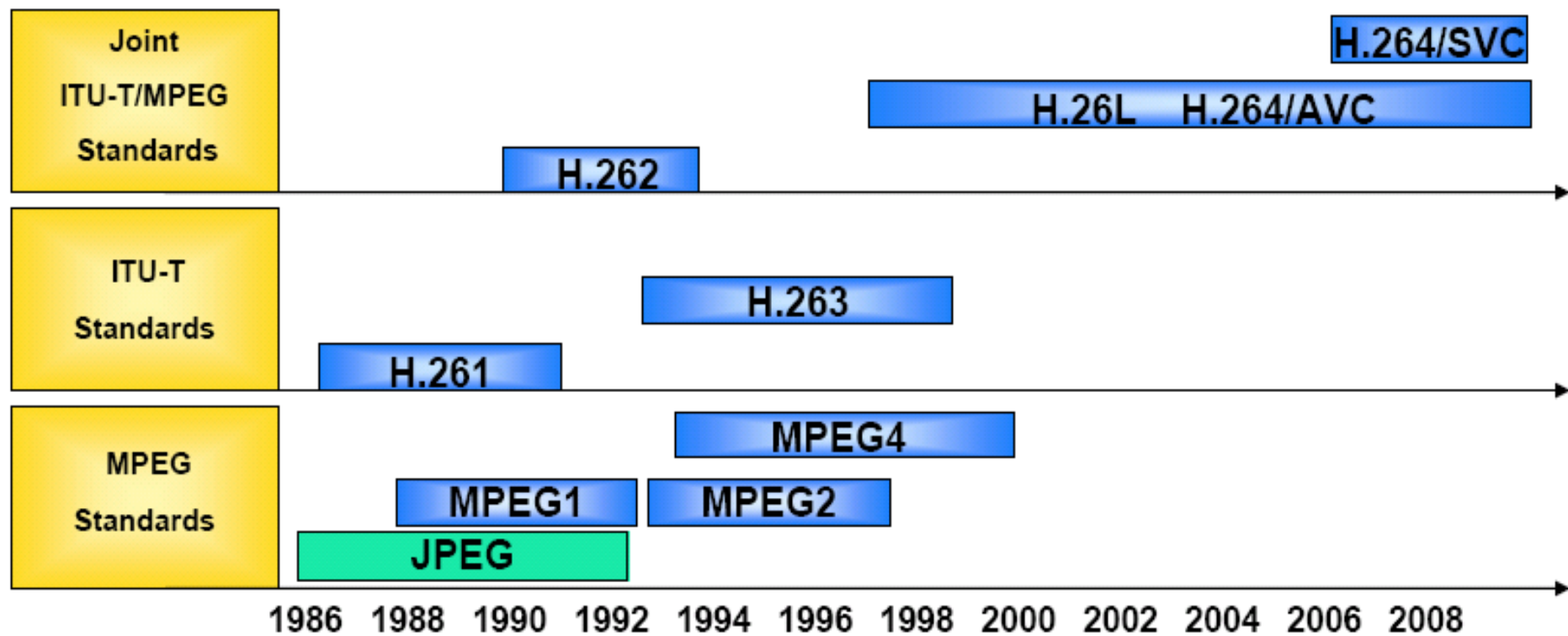
**Compression results in lower bit-rates which enables new applications and lower transmission and storage cost**

# [ Some Standards for Image/Video Compression ]

---

- **JPEG:** Joint Photographic Experts Group
- **ITU-T H.261:**  
Video codec for audiovisual services at px64Kbps
- **ITU-T H.263**  
Experts group on very low bit-rate video telephony
- **MPEG-1:** Moving Picture Experts Group  
Digital storage media, up to 1.5 Mbps
- **MPEG-2:**  
generic coding of moving pictures and associated audio
- **MPEG-4:**  
multiple bit-rate audiovisual coding, visual object coding

# Timeline of Codecs



# [ Data Compression ]

- The process of reducing the amount of data required to represent a given quality of information
- More efficient representation of digital images is necessary
- A compression system consists of two distinct structure of blocks:
  - Encoder (compressor) and decoder (decompressor)

# Compression/Coding

- Reducing the size of image data files
- Retain necessary information
- Lossless:
  - No data are lost
  - The original image can be reconstructed exactly
  - Image compression ratio: 2:1 to 3:1
- Lossy
  - Allow a loss in the actual image data
  - The original uncompressed image cannot be exactly recovered from the compressed file
  - Image compression ratio: 10:1 to 20:1

# Compression Ratio (Cr)

$$\begin{aligned} Cr &= \frac{\text{Source coder input size}}{\text{Source coder output size}} \\ &= \frac{\text{Uncompressed file size}}{\text{Compressed file size}} = \frac{SIZE_u}{SIZE_c} \end{aligned}$$

## ■ Example:

- Image of size 256x256 pixels, 256 grayscale levels  
compressed file size is 6554 bytes
- Original file size = 256x256(pixels) x 1(byte/pixel)  
= 65536 bytes
- The compression ratio is

# Bit per Pixel (BPP)

$$\text{BPP} = \frac{\text{Number of Bits}}{\text{Number of Pixels}} = \frac{8 \times (\text{Number of Bytes})}{\text{Number of Pixels}}$$

## ■ Example:

- Image of size 256x256 pixels, 256 grayscale levels  
compressed file size is 6554 bytes
- Original file size =  $256 \times 256 (\text{pixels}) \times 1 (\text{byte/pixel})$   
= 65536 bytes
- Compressed file =  $6554 (\text{bytes}) \times 8 (\text{bits/byte}) = 52432 \text{ bits}$
- Bits per Pixel =  $52432 / 65536 \approx 0.8$

# Fidelity Measure

- Fidelity measure is used to measure information loss
- Objective fidelity measure
  - Measured mathematically about the amount of error in the reconstructed image
- Subjective fidelity measure
  - Measured by human perception



# Objective Fidelity Measure

$f(x, y)$  = The  $N \times N$  original image

$\hat{f}(x, y)$  = The decompressed image

$$error(x, y) = \hat{f}(x, y) - f(x, y)$$

- **Peak Signal to Noise Ratio (PSNR)**

$$PSNR = 10 \times \log_{10} \left( \frac{255^2}{\frac{1}{mn} \sum_{x=1}^{m-1} \sum_{y=1}^{n-1} [\hat{f}(x, y) - f(x, y)]^2} \right)$$

Mean Squared Error (MSE)

- **Signal-to-noise ratio (SNR)**

$$SNR = 10 \times \log_{10} \left( \frac{f(x, y)^2}{\frac{1}{mn} \sum_{x=1}^{m-1} \sum_{y=1}^{n-1} [\hat{f}(x, y) - f(x, y)]^2} \right)$$

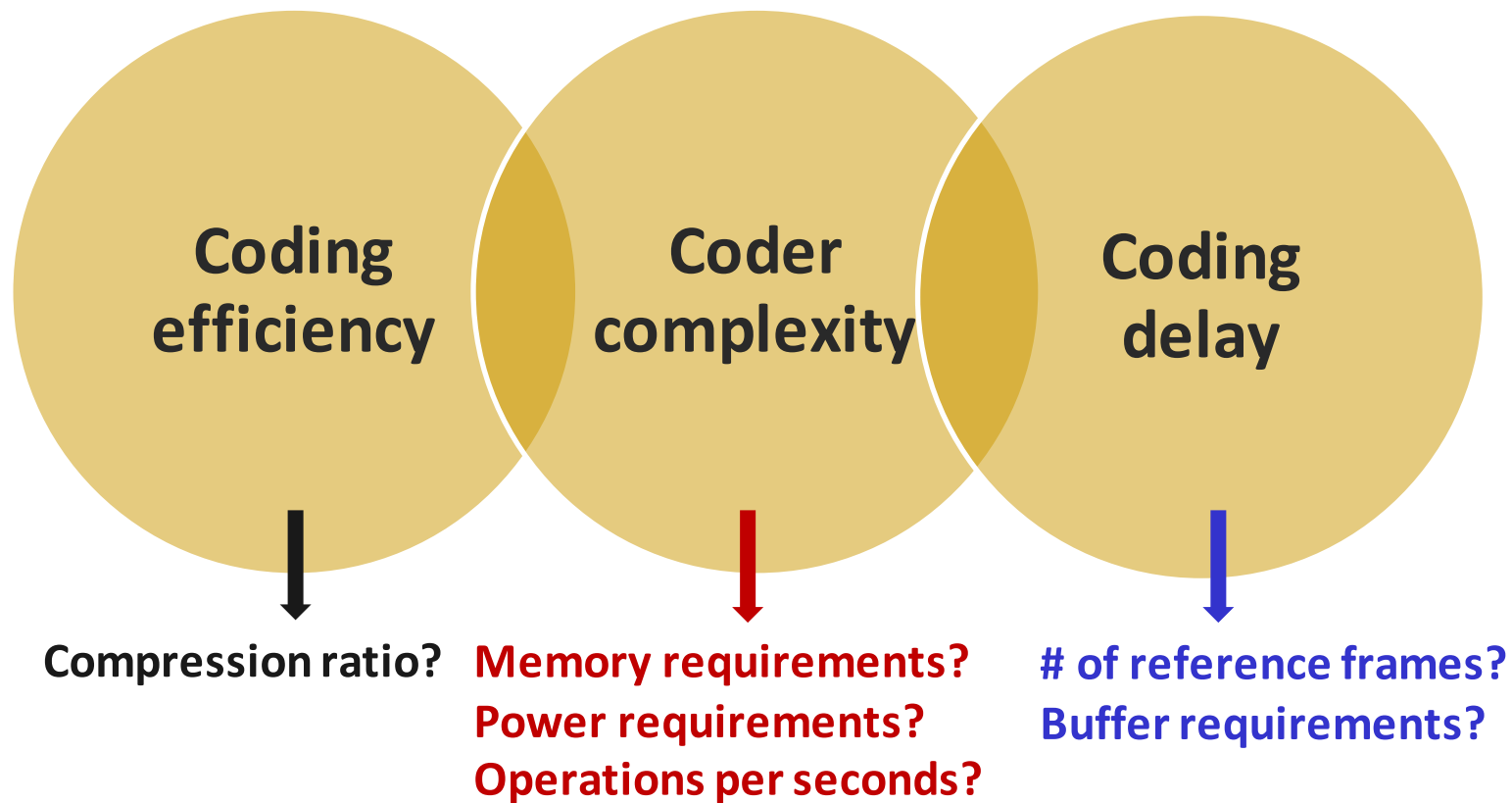
# Objective Fidelity Measure

- Root mean square error Rms

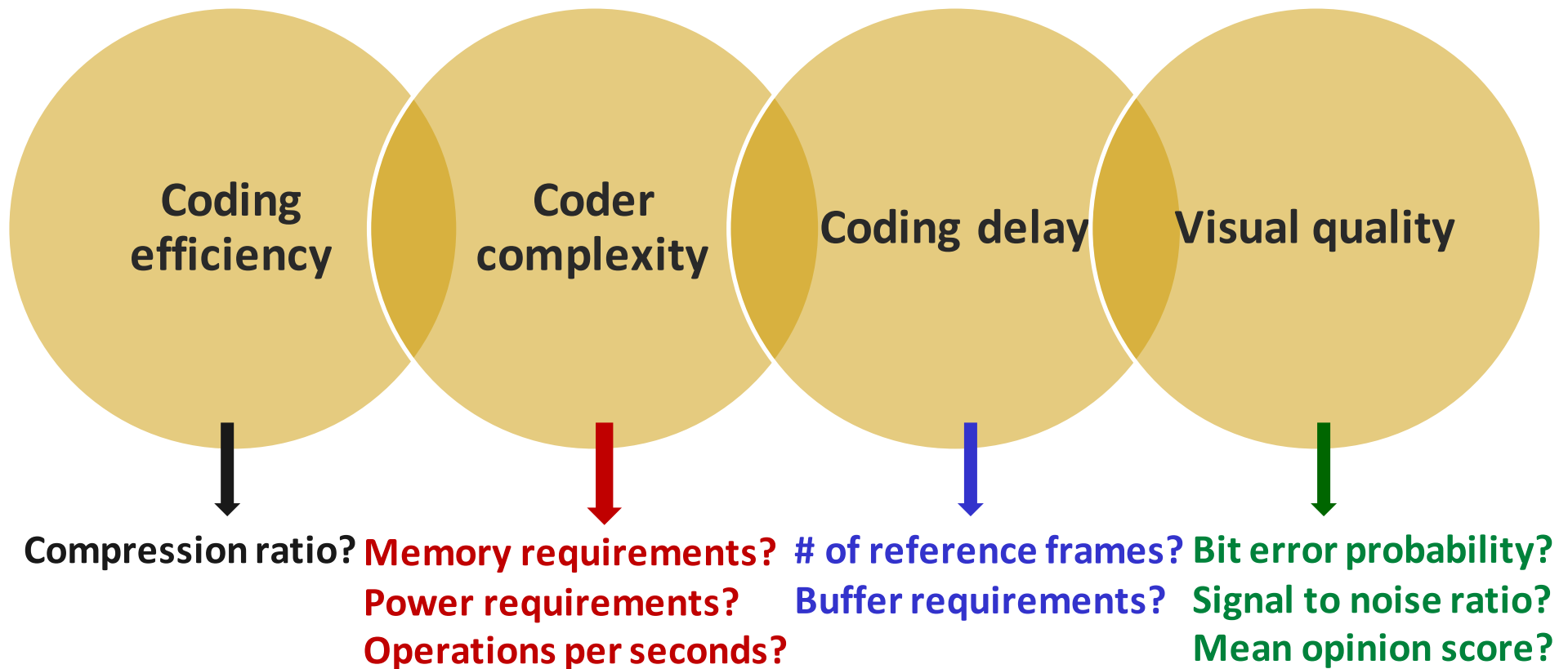
$$R_{ms} = \left[ \frac{1}{mn} \sum_{x=0}^{m-1} \sum_{y=0}^{n-1} [\hat{f}(x, y) - f(x, y)]^2 \right]^{1/2}$$

- Object fidelity measure
  - Easy to generate
  - Not necessarily correlated to our perception
  - Seemingly unbiased

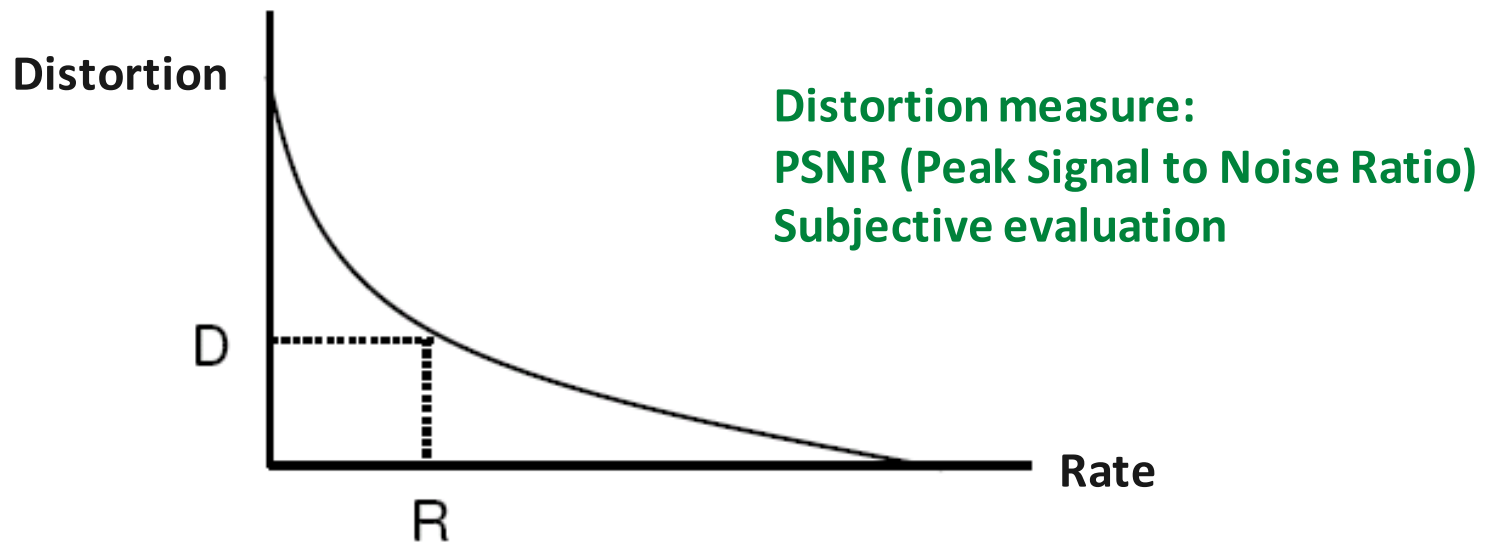
# [Tradeoffs in Lossless Compression]



# Tradeoffs in Lossy Compression



# Rate Distortion of Lossy Algorithms



- provide theoretical bounds for how much compression can be achieved using lossy compression methods
- determine the minimal amount of information  $R$  that should be transmitted over a channel, so that the input signal can be reconstructed at the receiver without exceeding a given distortion  $D$

# [ Rate Distortion ]

- A major branch of information theory
  - Address the problem of determining the minimal amount of information (or entropy)  $R$  that should be communicated over a channel
  - The source (input signal) can be approximately reconstructed at the receiver (output signal) without exceeding a given distortion  $D$
- What is the minimum bit rate required to encode a source while keeping the resulting degradation below a certain level ?

# [ Rate Distortion ]

- Rate-distortion function  $R(D)$  satisfies
  - For any given level of distortion  $D$ , it is possible to find a coding scheme with rate arbitrarily close to  $R(D)$  and average distortion arbitrarily close to  $D$
  - It is impossible to find a code that achieves reproduction with distortion  $D$  ( or better ) at a rate below  $R(D)$

# [ Rate Distortion ]

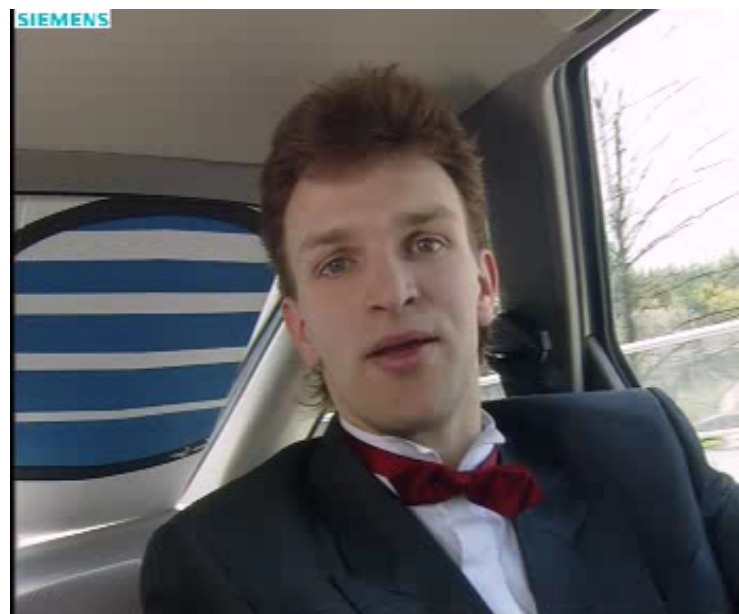
- Example
  - Carphone video  
10 fps, 6 seconds



Original



64Kbps



80  
256Kbps