

# Image Coding and JPEG



Trac D. Tran

ECE Department

The Johns Hopkins University

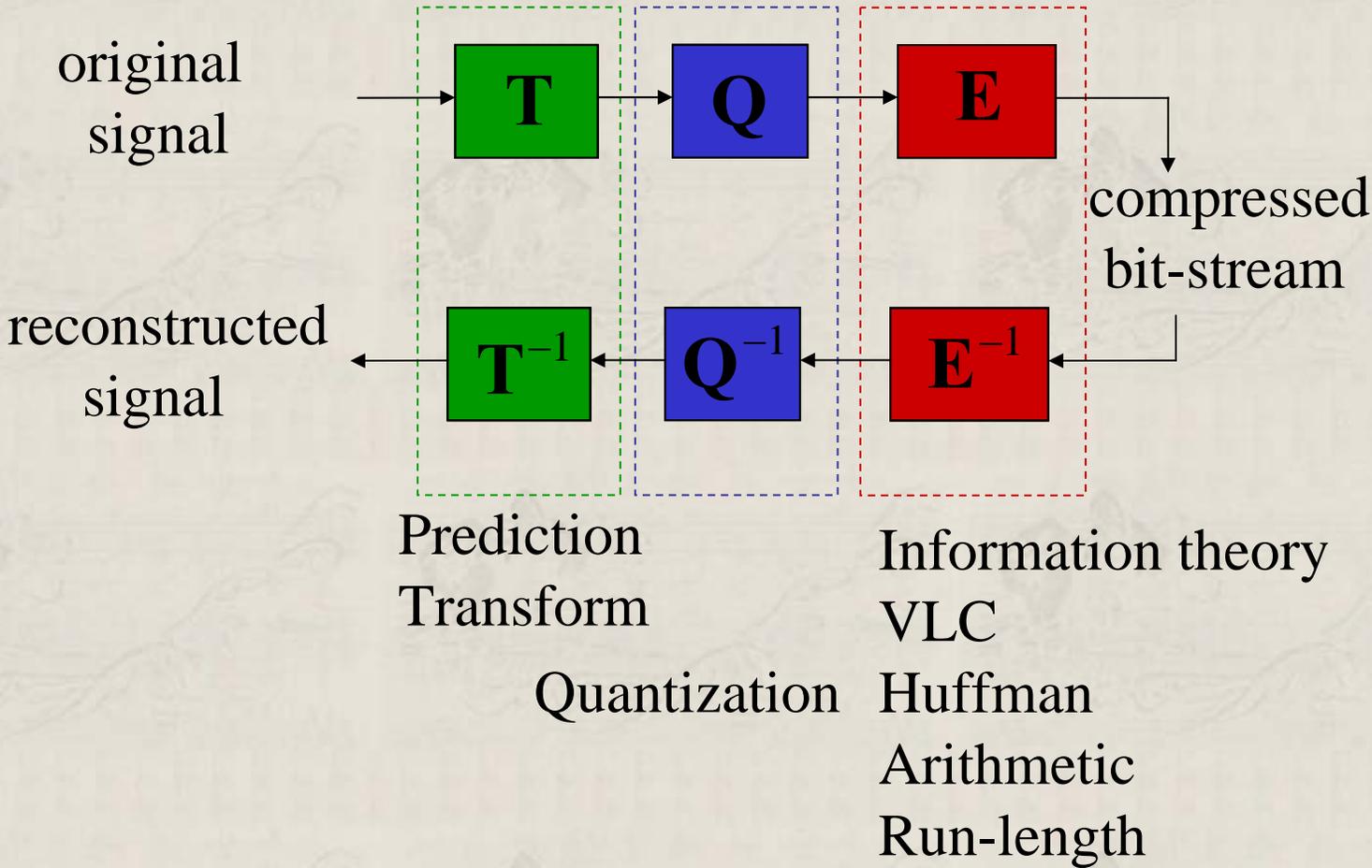
Baltimore MD 21218

# Outline

---

- ◆ Prediction
  - Open-loop differential pulse-code modulation (DPCM)
  - Closed-loop DPCM
  - Optimal linear prediction
- ◆ Transformation
  - Transform fundamentals
    - Basis functions, transform coefficients
    - Invertibility, unitary
    - 1D, 2D
  - Karhunen-Loeve Transform (KLT)
    - Optimal linear transform
  - Discrete Cosine Transform (DCT)
- ◆ Putting everything together: JPEG

# Reminder

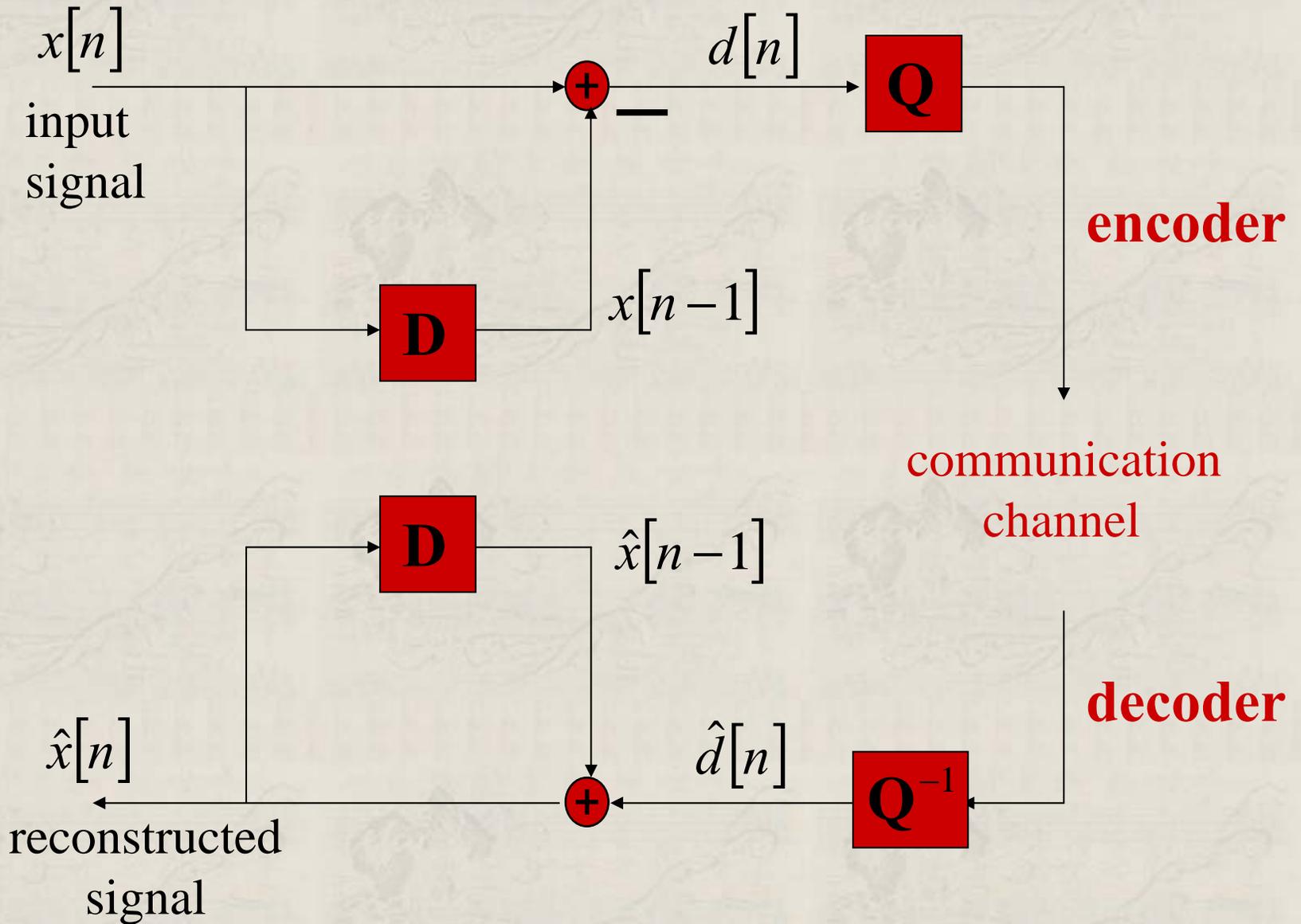


# Predictive Coding

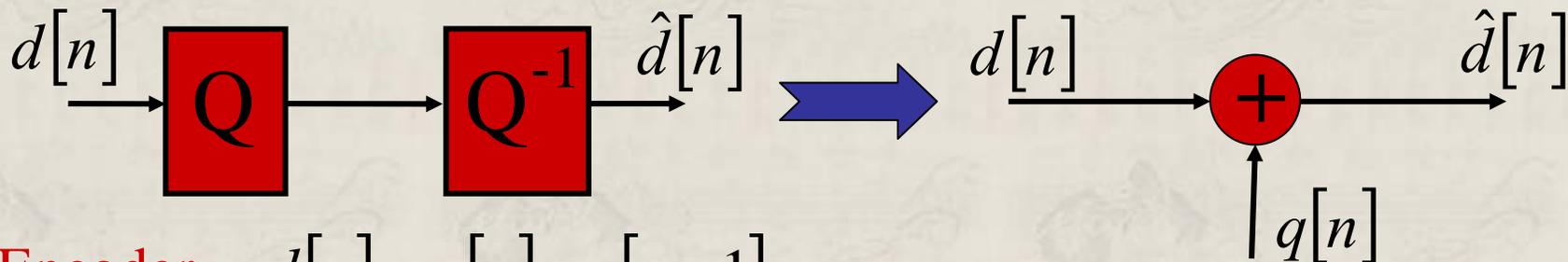
---

- ◆ We have only dealt with memory-less model so far
  - Each symbol/sample is quantized and/or coded without much knowledge on previous ones
- ◆ There is usually a strong correlation between neighboring symbols/samples in multimedia signals
- ◆ Simplest prediction scheme: take the difference!
- ◆ If the difference between two adjacent symbols/samples is quantized and encoded instead, we can achieve the same level of compression performance using fewer bits – the range of the differences should be a lot smaller.

# Open-Loop DPCM



# Open-Loop DPCM: Analysis



**Encoder**  $d[n] = x[n] - x[n-1]$

**Decoder**  $\hat{x}[n] = \hat{d}[n] + \hat{x}[n-1]$

$$\hat{d}[n] = d[n] + q[n]$$

$$\hat{d}[0] = d[0] + q[0]$$

$$\hat{x}[0] = \hat{d}[0] = d[0] + q[0] = x[0] + q[0]$$

$$\hat{d}[1] = d[1] + q[1]$$

$$\hat{x}[1] = \hat{d}[1] + \hat{x}[0] = d[1] + q[1] + x[0] + q[0]$$

$$= x[1] - x[0] + q[1] + x[0] + q[0]$$

$$= x[1] + q[0] + q[1]$$

**Quantization Error  
Accumulation**

$$\hat{x}[n] = x[n] + \sum_{i=0}^n q[i]$$



$$|e[n]| = |\hat{x}[n] - x[n]| = \left| \sum_{i=0}^n q[i] \right|$$

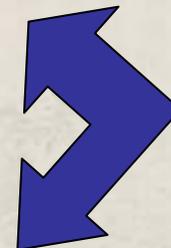
# Open-Loop DPCM: Analysis

- ◆ There seems to be a **model mismatch**

- **Encoder:**  $d[n] = x[n] - x[n-1]$

- **Decoder:**  $\hat{x}[n] = \hat{d}[n] + \hat{x}[n-1]$

$$\Rightarrow \hat{d}[n] = \hat{x}[n] - \hat{x}[n-1]$$



- ◆ How about using the reconstructed sample for the difference?

$$d[n] = x[n] - \hat{x}[n-1]$$

# Closed-Loop DPCM: Analysis

Modified prediction scheme  $d[n] = x[n] - \hat{x}[n-1]$

Decoder remains the same  $\hat{x}[n] = \hat{d}[n] + \hat{x}[n-1]$   
 $\overbrace{d[n] + q[n]}$

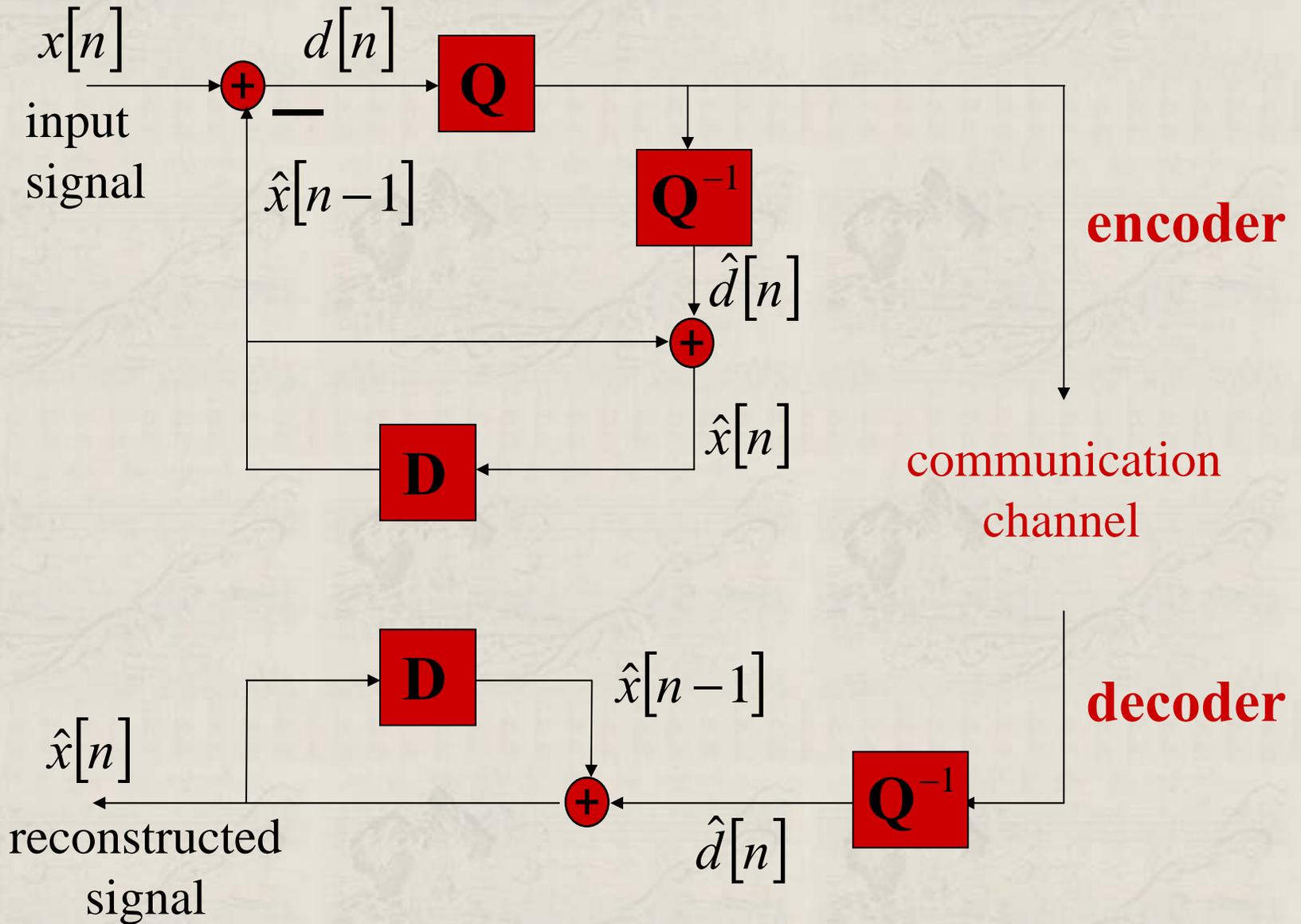
$$\hat{d}[0] = d[0] + q[0] \quad \hat{x}[0] = \hat{d}[0] = d[0] + q[0] = x[0] + q[0]$$

$$\begin{aligned} \hat{d}[1] &= d[1] + q[1] & \hat{x}[1] &= \hat{d}[1] + \hat{x}[0] = d[1] + q[1] + x[0] + q[0] \\ & & &= x[1] - \hat{x}[0] + q[1] + x[0] + q[0] \\ & & &= x[1] + q[1] \end{aligned}$$

$$\hat{x}[n] = x[n] + q[n]$$

**No error accumulation!**

# Closed-Loop DPCM

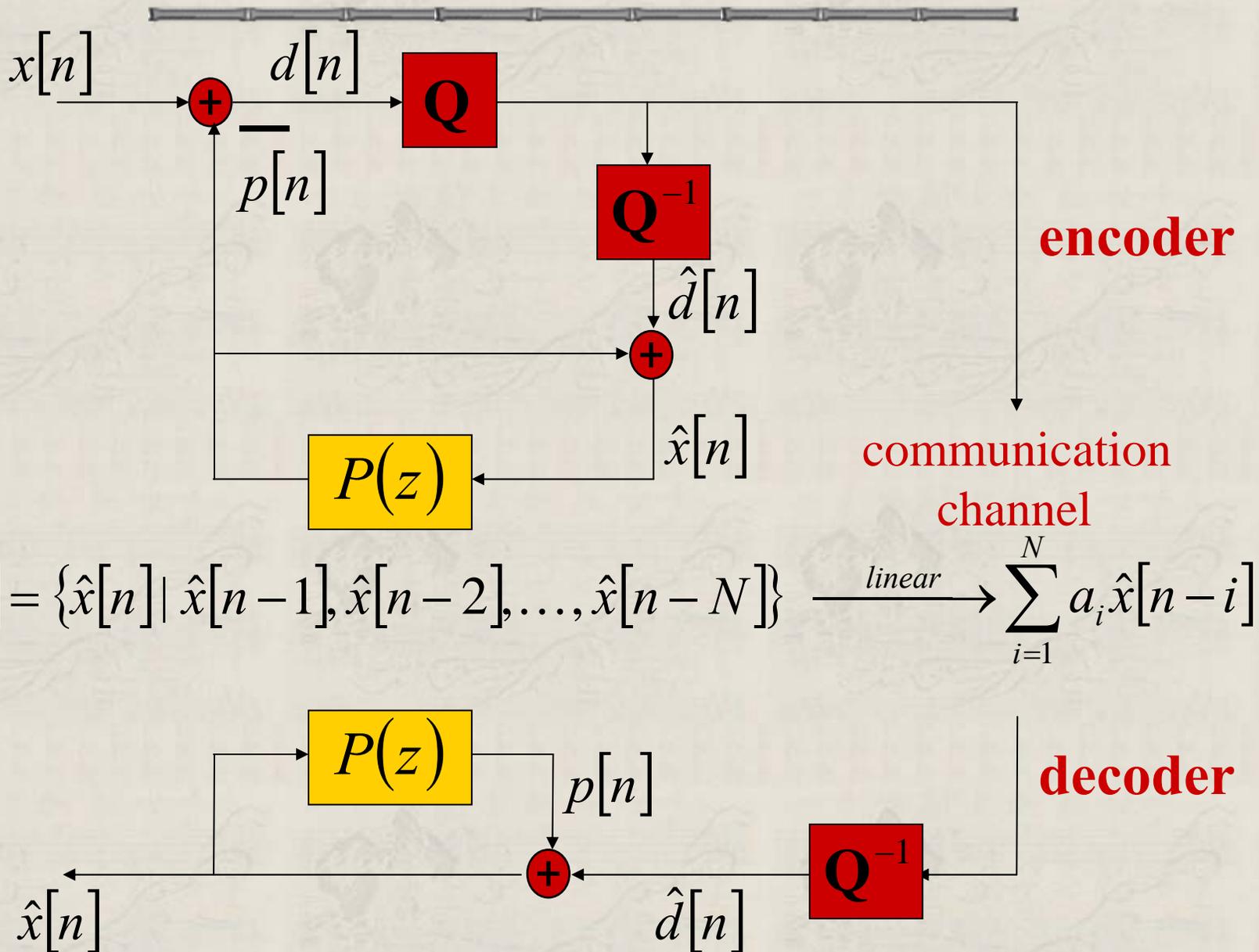


# Closed-Loop DPCM: Observations

---

- ◆ Quantization error does not accumulate
- ◆ Minor modification in prediction scheme leads to major encoder modification
  - Encoder now has decoder embedded inside
- ◆ Closed-loop & open-loop DPCM has the same decoder
- ◆ DPCM predicts current sample from last reconstructed one
- ◆ Generalization?
  - Replace the simple delay operator by more complicated & more sophisticated predictor  $P(z)$

# Linear Prediction



# Optimal Linear Prediction

## ◆ Problem

Find  $\{a_i\}$  s.t.  $D = \sigma_d^2 = E \left[ \left( x[n] - \sum_{i=1}^N a_i \hat{x}[n-i] \right)^2 \right]$  is minimized

## ◆ Assumptions

- Signal is WSS  $R_{xx}(k) = E[x[n]x[n+k]]$
- High bit rates, i.e., fine quantization

$$p[n] = \sum_{i=1}^N a_i \hat{x}[n-i] \approx \sum_{i=1}^N a_i x[n-i]$$

## ◆ Approach

$$\frac{\delta}{\delta a_i} D = 0$$

# Optimal Linear Prediction

$$\frac{\delta}{\delta a_i} D = \frac{\delta}{\delta a_i} E \left[ \left( x[n] - \sum_{i=1}^N a_i \hat{x}[n-i] \right)^2 \right] = 0$$

$$\frac{\delta}{\delta a_1} D = -2E \left[ \left( x[n] - \sum_{i=1}^N a_i \hat{x}[n-i] \right) x[n-1] \right] = 0$$

$$\frac{\delta}{\delta a_2} D = -2E \left[ \left( x[n] - \sum_{i=1}^N a_i \hat{x}[n-i] \right) x[n-2] \right] = 0$$

⋮

$$\frac{\delta}{\delta a_N} D = -2E \left[ \left( x[n] - \sum_{i=1}^N a_i \hat{x}[n-i] \right) x[n-N] \right] = 0$$

# Optimal Linear Prediction

$$E[x[n]x[n-1]] = \sum_{i=1}^N a_i E[x[n-i]x[n-1]] \Rightarrow R_{xx}(1) = \sum_{i=1}^N a_i R_{xx}(i-1)$$

$$E[x[n]x[n-2]] = \sum_{i=1}^N a_i E[x[n-i]x[n-2]] \Rightarrow R_{xx}(2) = \sum_{i=1}^N a_i R_{xx}(i-2)$$

⋮

⋮

$$E[x[n]x[n-N]] = \sum_{i=1}^N a_i E[x[n-i]x[n-N]] \Rightarrow R_{xx}(N) = \sum_{i=1}^N a_i R_{xx}(i-N)$$

# Optimal Linear Prediction

$$\begin{bmatrix} R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & & R_{xx}(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & \cdots & R_{xx}(0) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} R_{xx}(1) \\ R_{xx}(2) \\ \vdots \\ R_{xx}(N) \end{bmatrix}$$

Correlation matrix of WSS RP  $x[n]$

Cross correlation vector between  $x[n]$  and past observed samples

$$\mathbf{R}_{xx} \mathbf{a} = \mathbf{p} \quad \Rightarrow \quad \mathbf{a} = \mathbf{R}_{xx}^{-1} \mathbf{p}$$

approximated from time averaging

# Linear Signal Representation

input  
signal

transform  
coefficient

basis  
function

Synthesis

$$\mathbf{x} = \sum_{i=0}^{N-1} y_i \hat{\Phi}_i$$

Analysis

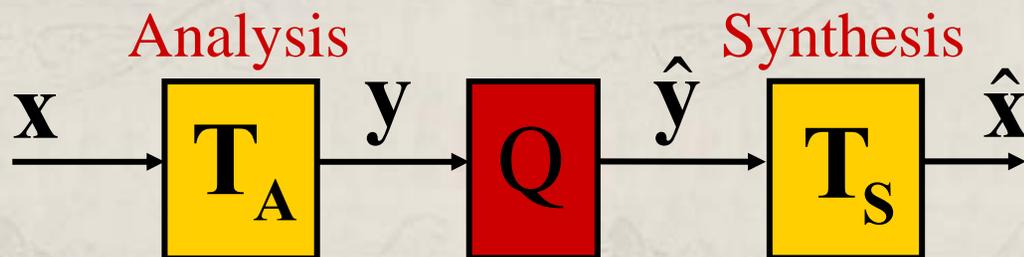
$$y_i = \langle \mathbf{x}, \Phi_i \rangle = \Phi_i^T \mathbf{x}$$

Approximation

$$\hat{\mathbf{x}} = \sum_{i=0}^{N-1} \hat{y}_i \hat{\Phi}_i$$

using as few  
coefficients  
as possible

# Transform Fundamentals



$$y_i = \langle \Phi_i, \mathbf{x} \rangle = \Phi_i^T \mathbf{x}$$

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} = \underbrace{\begin{bmatrix} \Phi_0^T \\ \Phi_1^T \\ \vdots \\ \Phi_{N-1}^T \end{bmatrix}}_{\mathbf{T}_A} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

- ◆ 1D Analysis Transform

$$\mathbf{y} = \mathbf{T}_A \mathbf{x}$$

$\{\Phi_i\}$  Analysis  
basis functions

- ◆ 1D Synthesis Transform

$$\hat{\mathbf{x}} = \mathbf{T}_S \hat{\mathbf{y}}$$

$\{\hat{\Phi}_i\}$  Synthesis  
basis functions

$$\begin{bmatrix} \hat{x}_0 \\ \hat{x}_1 \\ \vdots \\ \hat{x}_{N-1} \end{bmatrix} = \underbrace{\begin{bmatrix} \hat{\Phi}_0 & & & \\ & \hat{\Phi}_1 & & \\ & & \dots & \\ & & & \hat{\Phi}_{N-1} \end{bmatrix}}_{\mathbf{T}_S} \begin{bmatrix} \hat{y}_0 \\ \hat{y}_1 \\ \vdots \\ \hat{y}_{N-1} \end{bmatrix}$$

# Invertibility & Unitary

## ◆ Invertibility

- perfect reconstruction, bi-orthogonal, reversible

$$\mathbf{T}_S = \mathbf{T}_A^{-1} \Rightarrow \mathbf{T}_S \mathbf{T}_A = \mathbf{T}_A \mathbf{T}_S = \mathbf{I}$$
$$\langle \Phi_i, \hat{\Phi}_j \rangle = \delta[i - j]$$

## ◆ Unitary

- orthogonal, orthonormal

$$\mathbf{T}_S = \mathbf{T}_A^{-1} = \mathbf{T}_A^T \Rightarrow \mathbf{T}_A^T \mathbf{T}_A = \mathbf{T}_A \mathbf{T}_A^T = \mathbf{I}$$
$$\langle \Phi_i, \Phi_j \rangle = \delta[i - j]$$

same analysis & synthesis basis functions

# Norm Preservation

- ◆ Norm preservation property of orthonormal transform

$$\begin{aligned}\|y - \hat{y}\|_2 &= \|\mathbf{T}_a \mathbf{x} - \mathbf{T}_a \hat{\mathbf{x}}\|_2 = \|\mathbf{T}_a (\mathbf{x} - \hat{\mathbf{x}})\|_2 \\ &= \sqrt{\langle \mathbf{T}_a (\mathbf{x} - \hat{\mathbf{x}}), \mathbf{T}_a (\mathbf{x} - \hat{\mathbf{x}}) \rangle} = \sqrt{(\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{T}_a^T \mathbf{T}_a (\mathbf{x} - \hat{\mathbf{x}})} \\ &= \sqrt{(\mathbf{x} - \hat{\mathbf{x}})^T (\mathbf{x} - \hat{\mathbf{x}})} = \|\mathbf{x} - \hat{\mathbf{x}}\|_2\end{aligned}$$

- ◆ From a coding perspective
  - Q error in the transform domain equals Q error in the spatial domain!
  - Concentrate on the quantization of the transform coefficients

# 2D Separable Transformation

- ◆ 2D Analysis

$$\mathbf{y} = \mathbf{T}_A \mathbf{x} \mathbf{T}_A^T$$

$N \times N$       transforming rows

transforming columns

- ◆ 2D Synthesis

$$\mathbf{x} = \mathbf{T}_S \mathbf{y} \mathbf{T}_S^T = \mathbf{T}_S \mathbf{T}_A \mathbf{x} \mathbf{T}_A^T \mathbf{T}_S^T$$

$N \times N$

- ◆ 2D Orthogonal Synthesis

$$\mathbf{x} = \mathbf{T}_A^T \mathbf{y} \mathbf{T}_A = \mathbf{T}_A^T \mathbf{T}_A \mathbf{x} \mathbf{T}_A^T \mathbf{T}_A$$

# Example

- ◆ Discrete Fourier Transform (DFT)

$$F[k] = \sum_{n=0}^{N-1} f[n] W_N^{nk}$$

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k] W_N^{-nk}$$

$$n, k \in \{0, 1, \dots, N-1\} \quad W_N \equiv e^{-j\frac{2\pi}{N}}$$

$$\mathbf{T}_A = \left\{ W_N^{nk} \right\}$$

$$N = 4$$

$$\mathbf{T}_A = \begin{matrix} & \xrightarrow{n} & & & \\ \begin{matrix} \downarrow k \\ \left[ \begin{array}{cccc} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{array} \right] \end{matrix} & & & & \end{matrix}$$

$$\mathbf{T}_S = \frac{1}{N} \mathbf{T}_A^H$$

# KLT: Optimal Linear Transform

- ◆ Karhunen-Loeve Transform (KLT)
  - Hotelling transform, principle component analysis (PCA)
- ◆ Question:
  - Amongst the linear transforms, which one is the best decorrelator, offering the best “compression”?
- ◆ Problem:

Given an  $N$  - point signal  $\mathbf{x}$ . Find the set of orthonormal basis functions  $\{\Phi_i\}, i \in \{0, 1, \dots, N - 1\}$ , such that the MSE between the  $L$  - point truncated representation  $\hat{\mathbf{x}} = \sum_{i=0}^{L-1} y_i \Phi_i$  ( $L < N$ ) and the original signal  $\mathbf{x}$  is minimized.
- ◆ Assumptions:  $\mathbf{x}$  is zero-mean WSS RP.

# KLT

- ◆ Reminder:
  - Orthonormal constraint:  $\langle \Phi_i, \Phi_j \rangle = \Phi_i^T \Phi_j = \delta[i - j]$
  - Autocorrelation matrix

$$\mathbf{R}_{xx} = E[\mathbf{xx}^T] = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) & \cdots & R_{xx}(N-1) \\ R_{xx}(1) & R_{xx}(0) & & R_{xx}(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ R_{xx}(N-1) & R_{xx}(N-2) & \cdots & R_{xx}(0) \end{bmatrix}$$

$$\begin{aligned} MSE &\equiv E\left[\|\mathbf{x} - \hat{\mathbf{x}}\|^2\right] = E\left[\left\|\sum_{i=0}^{N-1} y_i \Phi_i - \sum_{i=0}^{L-1} y_i \Phi_i\right\|^2\right] = E\left[\left\|\sum_{i=L}^{N-1} y_i \Phi_i\right\|^2\right] \\ &= E\left[\left\langle \sum_{i=L}^{N-1} y_i \Phi_i, \sum_{j=L}^{N-1} y_j \Phi_j \right\rangle\right] = E\left[\left(\sum_{i=L}^{N-1} y_i \Phi_i^T\right) \left(\sum_{j=L}^{N-1} y_j \Phi_j\right)\right] \\ &= E\left[\sum_{i=L}^{N-1} |y_i|^2\right] = E\left[\sum_{i=L}^{N-1} \Phi_i^T \mathbf{xx}^T \Phi_i\right] = \sum_{i=L}^{N-1} \Phi_i^T E[\mathbf{xx}^T] \Phi_i \end{aligned}$$

# KLT

- ◆ Reminder:  $\mathbf{R}_{xx} \mathbf{e} = \lambda \mathbf{e}$   
eigenvector  $\nearrow$

eigenvalue

$$\frac{\delta}{\delta \mathbf{v}} [\mathbf{u}^T \mathbf{v}] = \mathbf{u}$$

$$\frac{\delta}{\delta \mathbf{v}} [\mathbf{v}^T \mathbf{A} \mathbf{v}] = 2 \mathbf{A} \mathbf{v}$$

$$\text{Minimize } MSE = \sum_{i=L}^{N-1} \Phi_i^T E[\mathbf{x} \mathbf{x}^T] \Phi_i = \sum_{i=L}^{N-1} \Phi_i^T \mathbf{R}_{xx} \Phi_i$$

$$\text{wrt } \langle \Phi_i, \Phi_j \rangle = \delta[i - j]$$

- ◆ Lagrange Multiplier

$$\frac{\delta}{\delta \Phi_i} \left[ \sum_{i=L}^{N-1} \Phi_i^T \mathbf{R}_{xx} \Phi_i - \lambda_i (\langle \Phi_i, \Phi_j \rangle - 1) \right] = 0$$

$$\frac{\delta}{\delta \Phi_i} \left[ \Phi_i^T \mathbf{R}_{xx} \Phi_i - \lambda_i (\langle \Phi_i, \Phi_i \rangle - 1) \right] = 0$$

$$2\mathbf{R}_{xx} \Phi_i - 2\lambda_i \Phi_i = 0 \Rightarrow \mathbf{R}_{xx} \Phi_i = \lambda_i \Phi_i \Rightarrow MSE = \sum_{i=L}^{N-1} \lambda_i$$

- ◆ Optimal coding scheme: send the larger eigenvalues first!

# KLT Problems

---

- ◆ KLT problems
  - Signal dependent
  - Computationally expensive
    - statistics need to be computed
    - no structure, no symmetry, no guarantee of stability
  - Real signals are really stationary
  - Encoder/Decoder communication
- ◆ Practical solutions
  - Assume a reasonable signal model
  - Blocking the signals to ensure stationary assumption holds
  - Making the transform matrix sparse & symmetric
  - Good KLT approximation for smooth signals: DCT!

# Reminder: Linear Signal Representation

input  
signal

transform  
coefficient

basis  
function

Representation

$$\mathbf{x} = \sum_{i=0}^N c_i \Psi_i$$

Decomposition

$$c_i = \langle \mathbf{x}, \Psi_i \rangle$$

Approximation

$$\hat{\mathbf{x}} = \sum_{i=0}^{L \ll N} c_i \Psi_i$$

using as few  
coefficients  
as possible

# Motivations

---

- ◆ Fundamental question: what is the best basis?
  - energy compaction: minimize a pre-defined error measure, say MSE, given  $L$  coefficients
  - maximize perceptual reconstruction quality
  - low complexity: fast-computable decomposition and reconstruction
  - intuitive interpretation
- ◆ How to construct such a basis? Different viewpoints!
- ◆ Applications
  - compression, coding
  - signal analysis
  - de-noising, enhancement
  - communications

# KLT: Optimal Linear Transform

$$\mathbf{R}_{\mathbf{xx}} \Phi_i = \lambda_i \Phi_i$$

$E[\mathbf{xx}^T]$  →

eigenvectors

$$KLT = \begin{bmatrix} \Phi_0 & \Phi_1 & \dots & \Phi_{N-1} \end{bmatrix}$$

- ◆ Signal dependent
- ◆ Require stationary signals
- ◆ How do we communicate bases to the decoder?
- ◆ How do we design “good” signal-independent transform?

# Discrete Cosine Transforms

- ◆ Type I

$$K_i = \begin{cases} 1/\sqrt{2}, & i = 0, M \\ 1, & \text{otherwise} \end{cases}$$

$$[C^I] = \sqrt{\frac{2}{M}} \left[ K_m K_n \cos\left(\frac{mn\pi}{M}\right) \right], \quad m, n \in \{0, 1, \dots, M\}$$

- ◆ Type II

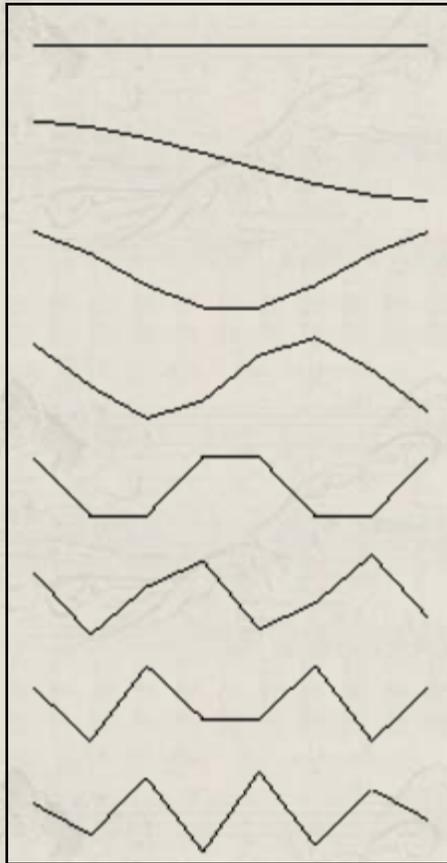
$$[C^{II}] = \sqrt{\frac{2}{M}} \left[ K_m \cos\left(\frac{m(n+1/2)\pi}{M}\right) \right], \quad m, n \in \{0, 1, \dots, M-1\}$$

- ◆ Type III
- $$[C^{III}] = \sqrt{\frac{2}{M}} \left[ K_n \cos\left(\frac{(m+1/2)n\pi}{M}\right) \right], \quad m, n \in \{0, 1, \dots, M-1\}$$

$$[C^{IV}] = \sqrt{\frac{2}{M}} \left[ \cos\left(\frac{(m+1/2)(n+1/2)\pi}{M}\right) \right], \quad m, n \in \{0, 1, \dots, M-1\}$$

# DCT Type-II

DCT basis



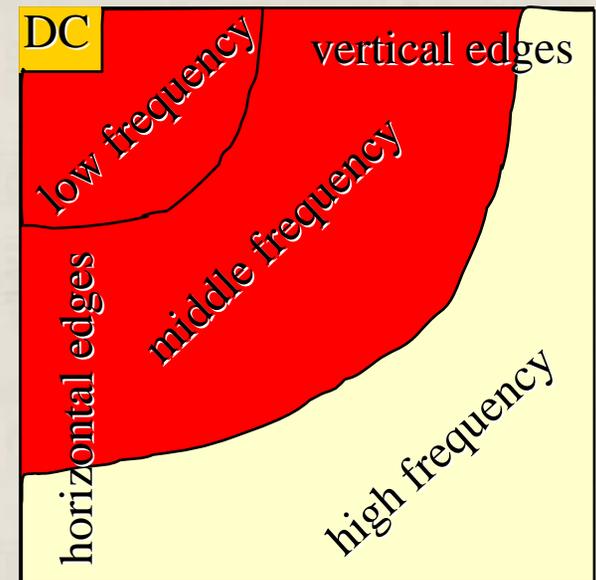
$$\begin{cases} X[m] = \sqrt{\frac{2}{M}} K_m \sum_{n=0}^{M-1} x[n] \cos \left[ \frac{(2n+1)m\pi}{2M} \right] \\ x[n] = \sqrt{\frac{2}{M}} K_n \sum_{m=0}^{M-1} X[m] \cos \left[ \frac{(2m+1)n\pi}{2M} \right] \end{cases}$$

$$m, n = 0, 1, \dots, M-1$$

$$K_i = \begin{cases} \frac{1}{\sqrt{2}}, & i = 0 \\ 1, & i \neq 0 \end{cases}$$

- orthogonal
- real coefficients
- symmetry
- near-optimal
- fast algorithms

8 x 8 block



# DCT Symmetry

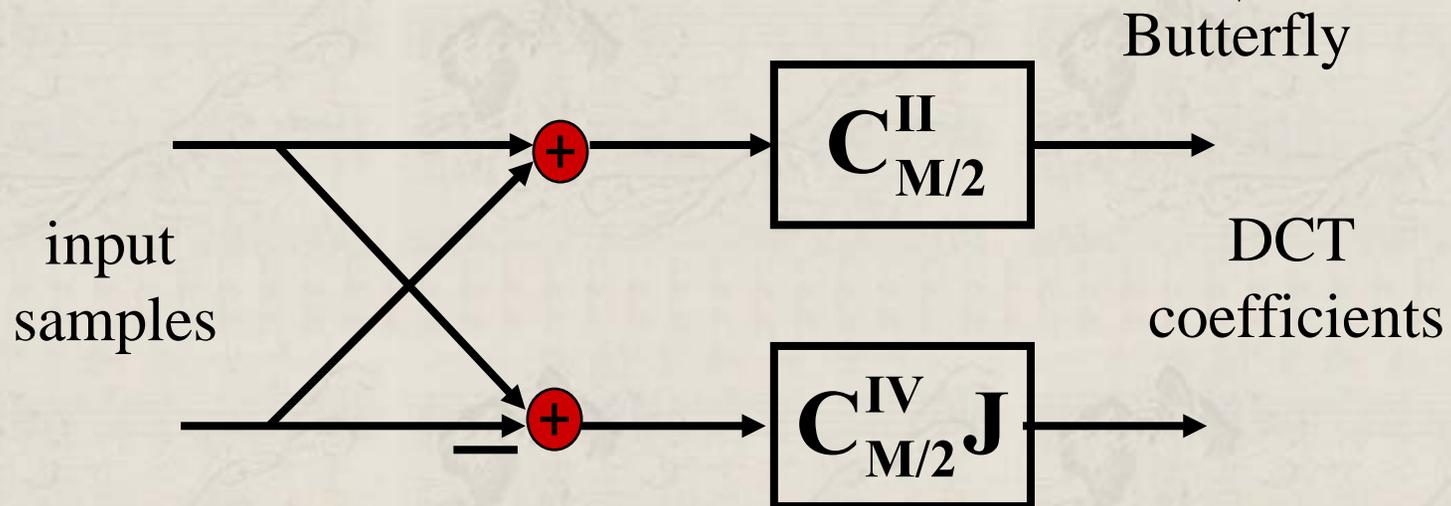
$$\begin{aligned} & \cos\left(\frac{m(2(M-1-n)+1)\pi}{2M}\right) \\ &= \cos\left(\frac{(2M-2-2n+1)m\pi}{2M}\right) \\ &= \cos\left[\frac{2Mm\pi}{2M} - \frac{(2n+1)m\pi}{2M}\right] \\ &= \pm \cos\left[\frac{(2n+1)m\pi}{2M}\right] \end{aligned}$$

DCT basis functions  
are either symmetric  
or anti-symmetric

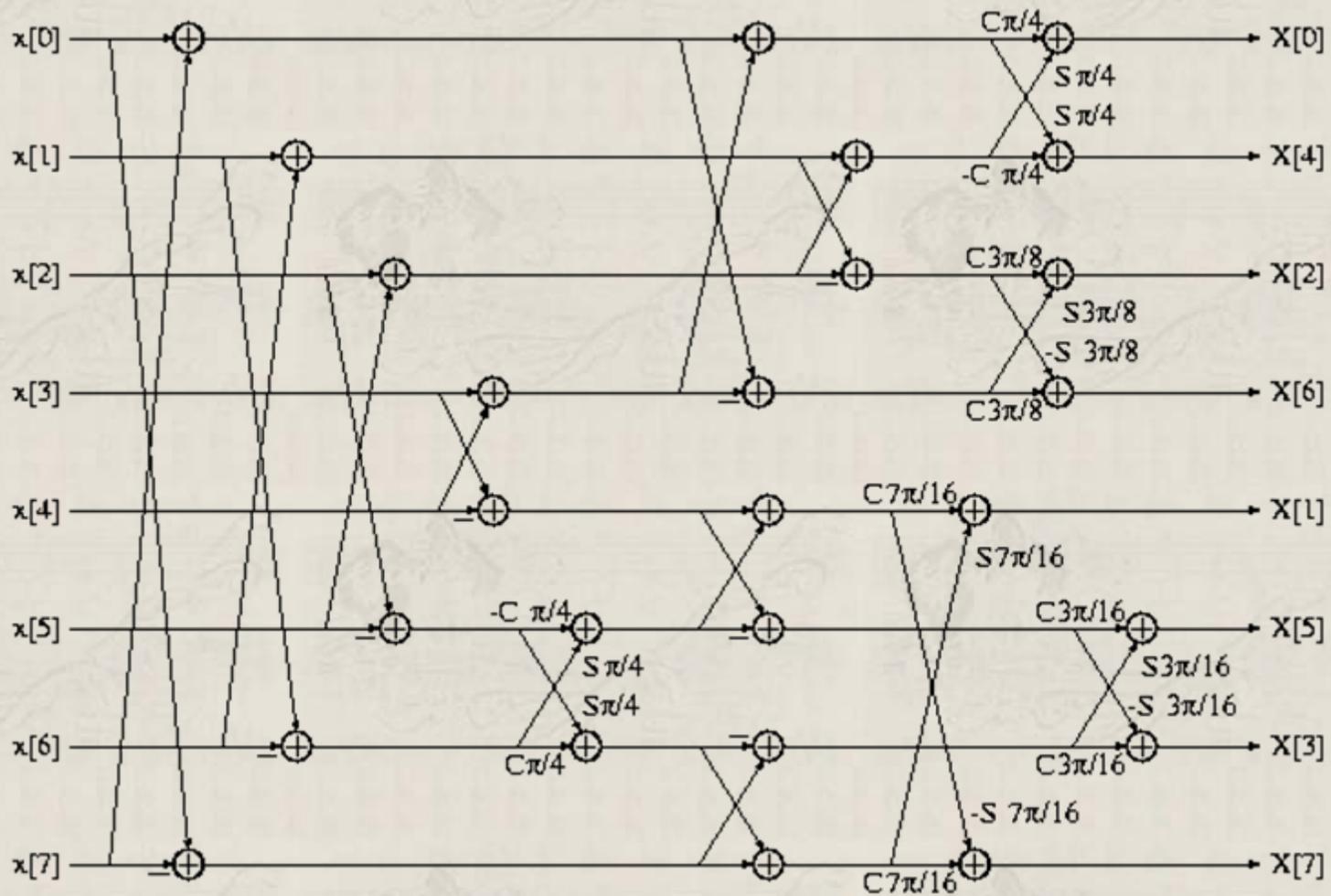
# DCT: Recursive Property

- An M-point DCT-II can be implemented via an M/2-point DCT-II and an M/2-point DCT-IV

$$[C_M^{II}] = \frac{1}{\sqrt{2}} \begin{bmatrix} C_{M/2}^{II} & \mathbf{0} \\ \mathbf{0} & C_{M/2}^{IV} J \end{bmatrix} \begin{bmatrix} I & J \\ J & -I \end{bmatrix}$$



# Fast DCT Implementation



13 multiplications and 29 additions per 8 input samples

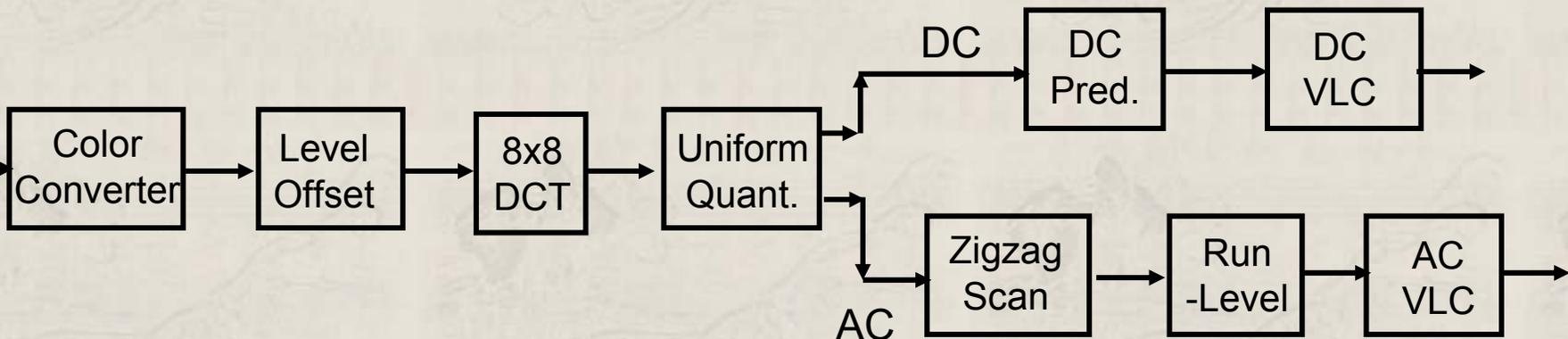
# Block DCT

$$\begin{bmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \vdots \\ \mathbf{X}_N \end{bmatrix} = \begin{bmatrix} \mathbf{C}_M^{\text{II}} & \mathbf{0} & & \\ \mathbf{0} & \mathbf{C}_M^{\text{II}} & \mathbf{0} & \\ & \mathbf{0} & \mathbf{C}_M^{\text{II}} & \mathbf{0} \\ & & \mathbf{0} & \mathbf{C}_M^{\text{II}} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix}$$

output blocks  
of DCT coefficients,  
each of size M

input blocks,  
each of size M

# Overall Structure of JPEG



- ◆ Color converter: RGB to YUV
- ◆ Level offset: subtract  $2^{(N-1)}$ . N: bits / pixel.
- ◆ Quantization: Different step size for different coefficients
- ◆ DC: Predict from DC of previous block
- ◆ AC:
  - Zigzag scan to get 1-D data
  - Run-level: joint coding of non-zero coeffs and number of zeros before it.

# JPEG Quantization

- ◆ Uniform mid-tread quantizer
- ◆ Larger step sizes for chroma components
- ◆ Different coefficients have different step sizes
  - Smaller steps for low frequency coefficients (more bits)
  - Larger steps for high frequency coefficients (less bits)
  - Human visual system is not sensitive to error in high frequency.

◆ Luma Quantization Table

|    |    |    |    |     |     |     |     |
|----|----|----|----|-----|-----|-----|-----|
| 16 | 11 | 10 | 16 | 24  | 40  | 51  | 51  |
| 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  |

◆ Chroma Quantization Table

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| 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 |

- ◆ Actual step size: Scale the basic table by a quality factor.

# Scaling of Quantization Table

◆ Actual Q table = scaling x Basic Q table:

- quality factor  $\leq 50$ : scaling = 50 / quality;
- quality factor  $> 50$ : scaling = 2 - quality / 50;

|    |    |    |    |     |     |     |
|----|----|----|----|-----|-----|-----|
| 16 | 11 | 10 | 16 | 24  | 40  | 51  |
| 12 | 12 | 14 | 19 | 26  | 58  | 60  |
| 14 | 13 | 16 | 24 | 40  | 57  | 69  |
| 14 | 17 | 22 | 29 | 51  | 87  | 80  |
| 18 | 22 | 37 | 56 | 68  | 109 | 103 |
| 24 | 35 | 55 | 64 | 81  | 104 | 113 |
| 49 | 64 | 78 | 87 | 103 | 121 | 120 |
| 72 | 92 | 95 | 98 | 112 | 100 | 103 |

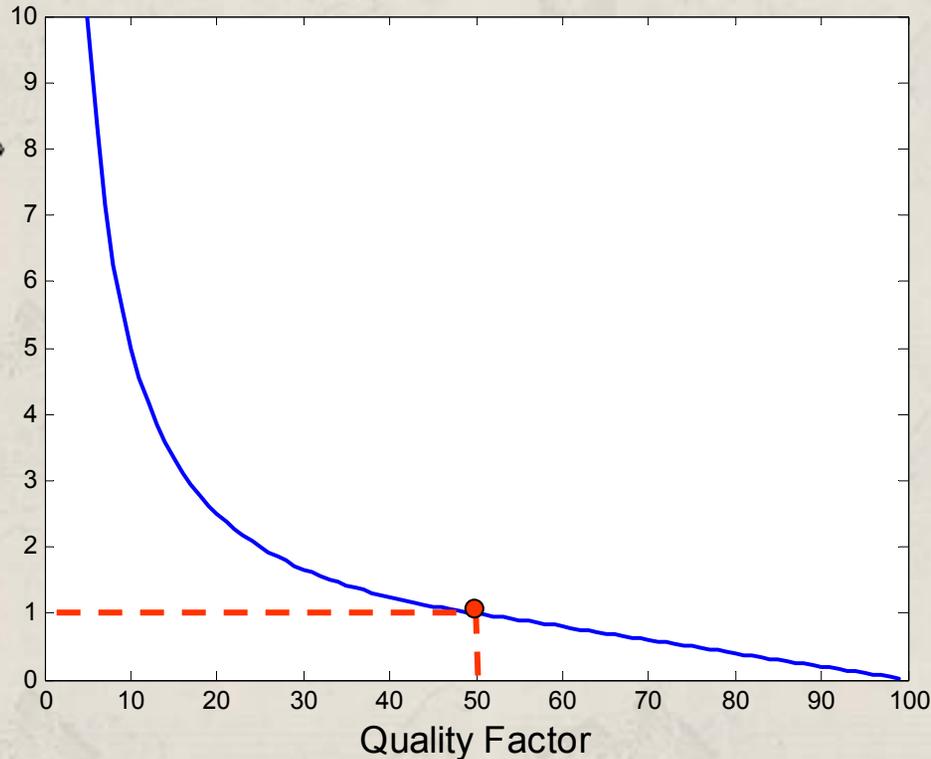
| Quality Factor | Scaling |
|----------------|---------|
|----------------|---------|

|    |     |
|----|-----|
| 10 | 5.0 |
|----|-----|

|    |     |
|----|-----|
| 20 | 2.5 |
|----|-----|

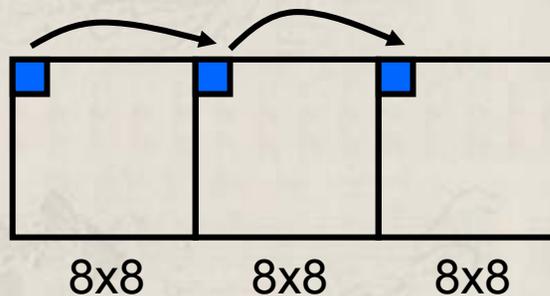
|    |     |
|----|-----|
| 50 | 1.0 |
|----|-----|

|    |     |
|----|-----|
| 75 | 0.5 |
|----|-----|



# DC Prediction

- ◆ DC Coefficients: average of a block
- ◆ DC of neighboring blocks are still similar to each others: redundancy
- ◆ The redundancy can be removed by differential coding:
  - $e(n) = DC(n) - DC(n-1)$
- ◆ Only encode the prediction error  $e(n)$



DC coeffs  
of Lena

# Coefficient Category

- ◆ Divide coefficients into categories of **exponentially increased sizes**
- ◆ Use Huffman code to encode category ID
- ◆ Use **fixed length code within each category**
- ◆ Similar to Exponential Golomb code

| Ranges                           | Range Size | DC Cat. ID | AC Cat. ID |
|----------------------------------|------------|------------|------------|
| 0                                | 1          | 0          | N/A        |
| -1, 1                            | 2          | 1          | 1          |
| -3, -2, 2, 3                     | 4          | 2          | 2          |
| -7, -6, -5, -4, 4, 5, 6, 7       | 8          | 3          | 3          |
| -15, ..., -8, 8, ..., 15         | 16         | 4          | 4          |
| -31, ..., -16, 16, ..., 31       | 32         | 5          | 5          |
| -63, ..., -32, 32, ..., 63       | 64         | 6          | 6          |
| ...                              | ...        | ...        | ...        |
| [-32767, -16384], [16384, 32767] | 32768      | 15         | 15         |

# Coding of DC Coefficients

- ◆ Encode  $e(n) = DC(n) - DC(n-1)$

| DC Cat. | Prediction Errors          | Base Codeword |
|---------|----------------------------|---------------|
| 0       | 0                          | 010           |
| 1       | -1, 1                      | 011           |
| 2       | -3, -2, 2, 3               | 100           |
| 3       | -7, -6, -5, -4, 4, 5, 6, 7 | 00            |
| 4       | -15, ..., -8, 8, ..., 15   | 101           |
| 5       | -31, ..., -16, 16, ..., 31 | 110           |
| 6       | -63, ..., -32, 32, ..., 63 | 1110          |
| ...     | ...                        | ...           |

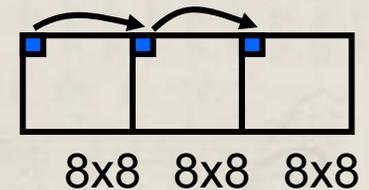
Our example:

DC: 8. Assume last DC: 5

Cat.: 2, index 3

$$\rightarrow e = 8 - 5 = 3.$$

$\rightarrow$  Bitstream: 10011







# Coding of AC Coefficients

- ◆ Many AC coefficients are zeros:
  - Huffman coding is not efficient for symbol with prob.  $> 1/2$
- ◆ **Run-level coding**: Jointly encode a non-zero coefficient and the number of zeros **before** it (**run of zeros**): (**run**, **level**) event
- ◆ **Disadvantage**: Symbol set is enlarged: #Run x #Level
- ◆ **Tradeoff**:
  - Run: encode up to 15 zeros. Apply **escape coding** for greater values.
  - Level: Divide level into 16 categories, as in DC.
  - Apply Huffman coding to the joint **Run / Category** event:
    - Max symbol set size:  $16 \times 16 = 256$ .
  - Followed by fixed length code to signal the level index within each category
- ◆ **Example**: zigzag scanning result  
24 -31 0 -4 -2 0 6 -12 0 0 0 -1 -1 0 0 0 2 -2 0 0 0 0 0 -1 EOB
- ◆ (Run, level) representation:
- ◆ (0, 24), (0, -31), (1, -4), (0, -2), (1, 6), (0, -12), (3, -1), (0, -1), (3, 2), (0, -2), (5, -1), EOB

# Coding of AC Coefficients

| Run / Cat. | Base codeword | Run / Cat. | Base Codeword  | ... | Run / Cat. | Base codeword       |
|------------|---------------|------------|----------------|-----|------------|---------------------|
| <b>EOB</b> | 1010          | -          | -              | ... | <b>ZRL</b> | 1111 1111 001       |
| 0/1        | 00            | 1/1        | 1100           | ... | 15/1       | 1111 1111 1111 0101 |
| 0/2        | 01            | 1/2        | 11011          | ... | 15/2       | 1111 1111 1111 0110 |
| 0/3        | 100           | <b>1/3</b> | <b>1111001</b> | ... | 15/3       | 1111 1111 1111 0111 |
| 0/4        | 1011          | 1/4        | 111110110      | ... | 15/4       | 1111 1111 1111 1000 |
| <b>0/5</b> | <b>11010</b>  | 1/5        | 11111110110    | ... | 15/5       | 1111 1111 1111 1001 |
| ...        | ...           | ...        | ...            | ... | ...        | ...                 |

- ◆ ZRL: represent 16 zeros when number of zeros exceeds 15.
    - Example: 20 zeros followed by -1: (ZRL), (4, -1).
  - ◆ (Run, Level) sequence: (0, 24), (0, -31), (1, -4), .....
  - ◆ Run/Cat. Sequence: 0/5, 0/5, 1/3, ...
- 24 is the 24-th entry in Category 5 → (0, 24): **11010** 11000
- 4 is the 3-th entry in Category 3 → (1, -4): **1111001** 011

# A Complete Example

◆ Original data:

2-D DCT

|     |     |     |     |     |     |     |     |        |      |      |      |      |      |      |      |
|-----|-----|-----|-----|-----|-----|-----|-----|--------|------|------|------|------|------|------|------|
| 124 | 125 | 122 | 120 | 122 | 119 | 117 | 118 | 39.8   | 6.5  | -2.2 | 1.2  | -0.3 | -1.0 | 0.7  | 1.1  |
| 121 | 121 | 120 | 119 | 119 | 120 | 120 | 118 | -102.4 | 4.5  | 2.2  | 1.1  | 0.3  | -0.6 | -1.0 | -0.4 |
| 126 | 124 | 123 | 122 | 121 | 121 | 120 | 120 | 37.7   | 1.3  | 1.7  | 0.2  | -1.5 | -2.2 | -0.1 | 0.2  |
| 124 | 124 | 125 | 125 | 126 | 125 | 124 | 124 | -5.6   | 2.2  | -1.3 | -0.8 | 1.4  | 0.2  | -0.1 | 0.1  |
| 127 | 127 | 128 | 129 | 130 | 128 | 127 | 125 | -3.3   | -0.7 | -1.7 | 0.7  | -0.6 | -2.6 | -1.3 | 0.7  |
| 143 | 142 | 143 | 142 | 140 | 139 | 139 | 139 | 5.9    | -0.1 | -0.4 | -0.7 | 1.9  | -0.2 | 1.4  | 0.0  |
| 150 | 148 | 152 | 152 | 152 | 152 | 150 | 151 | 3.9    | 5.5  | 2.3  | -0.5 | -0.1 | -0.8 | -0.5 | -0.1 |
| 156 | 159 | 158 | 155 | 158 | 158 | 157 | 156 | -3.4   | 0.5  | -1.0 | 0.8  | 0.9  | 0.0  | 0.3  | 0.0  |

◆ Quantized by basic table:

|    |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|
| 2  | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| -9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3  | 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 |

Q table:

|    |     |     |
|----|-----|-----|
| 16 | 11  | ... |
| 12 | ... |     |
| 14 | ... |     |

$$\text{floor}(39.8/16 + 0.5) = 2$$

$$\text{floor}(6.5/11 + 0.5) = 1$$

$$-\text{floor}(102.4/12 + 0.5) = -9$$

$$\text{floor}(37.7/14 + 0.5) = 3$$

◆ Zigzag scanning:

2 1 -9 3 EOB

# A Complete Example

- ◆ Zigzag scanning:

2 1 -9 3 EOB

- ◆ Show details of encoding:

- ◆ Dequantization:

|      |    |   |   |   |   |   |   |
|------|----|---|---|---|---|---|---|
| 32   | 11 | 0 | 0 | 0 | 0 | 0 | 0 |
| -108 | 0  | 0 | 0 | 0 | 0 | 0 | 0 |
| 42   | 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 |

- ◆ Reconstructed block:

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| 122 | 122 | 121 | 121 | 120 | 119 | 119 | 118 |
| 121 | 121 | 120 | 119 | 119 | 118 | 117 | 117 |
| 120 | 120 | 120 | 119 | 118 | 117 | 117 | 117 |
| 123 | 123 | 122 | 122 | 121 | 120 | 120 | 120 |
| 131 | 130 | 130 | 129 | 128 | 128 | 127 | 127 |
| 142 | 141 | 141 | 140 | 139 | 139 | 138 | 138 |
| 153 | 152 | 152 | 151 | 150 | 150 | 149 | 149 |
| 159 | 159 | 159 | 158 | 157 | 157 | 156 | 156 |

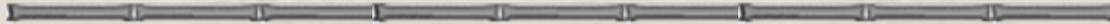
- ◆ MSE: 5.67

# Progressive JPEG

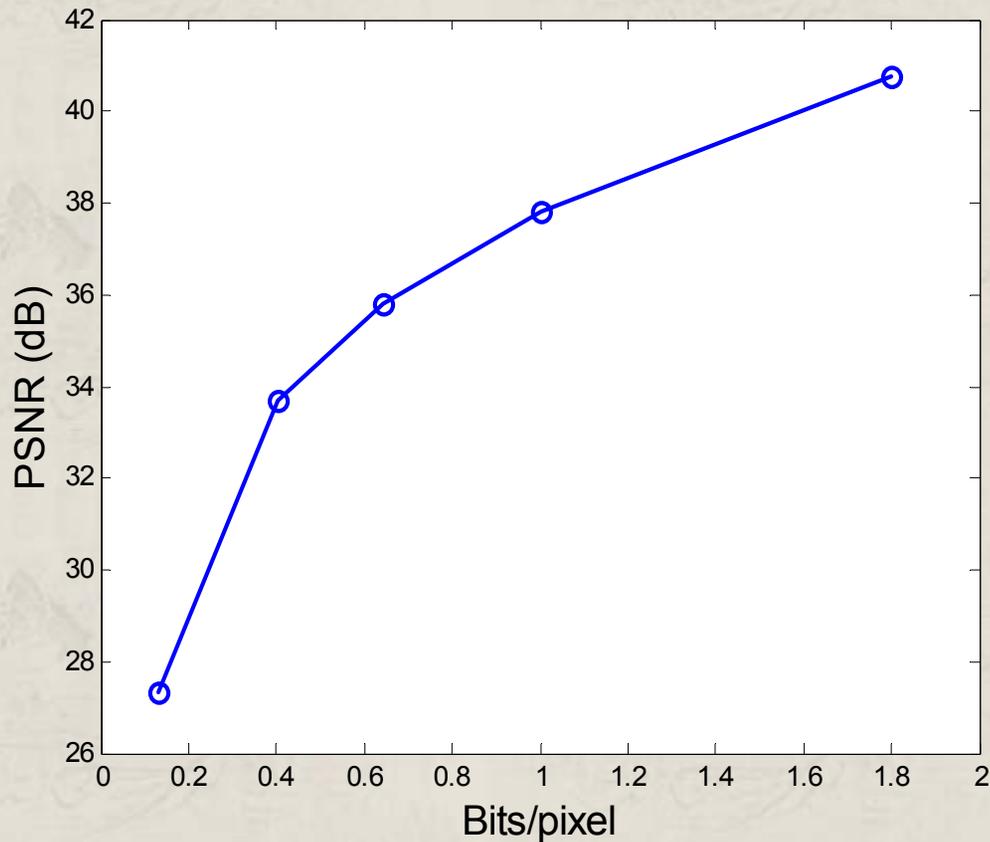
---

- ◆ Baseline JPEG encodes the image block by block:
  - Decoder has to wait till the end to decode and display the entire image.
- ◆ Progressive: Coding DCT coefficients in multiple scans
  - The first scan generates a low-quality version of the entire image
  - Subsequent scans refine the entire image gradually.
- ◆ Two procedures defined in JPEG:
  - Spectral selection:
    - Divide all DCT coefficients into several bands (low, middle, high frequency subbands...)
    - Bands are coded into separate scans
  - Successive approximation:
    - Send MSB of all coefficients first.
    - Send lower significant bits in subsequent scans.

# JPEG Coding Result for Lena



Lena



Quality factor:

5      25      50      75      90

QF  
25



QF  
5



Blocking artifact →



# SPIHT: Sorting Pass 2

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

- ◆ T=8
- ◆ (1,1)D significant? Yes
- ◆ (1,2) significant? No
- ◆ (2,1) significant? No
- ◆ (2,2) significant? No
- ◆ LIP = { (1,2), (2,1), (2,2) }. LIS = { (1,1)L }
- ◆ (1,1)L significant? Yes
- ◆ LIS = { (1,2)D, (2,1)D, (2,2)D }
- ◆ Is (1,2)D significant? Yes
- ◆ Is (1,3) significant? Yes
- ◆ LSP = { (1,1), (1,3) }
- ◆ Is (2,3) significant? Yes
- ◆ LSP = { (1,1), (1,3), (2,3) }

1

0

0

0

1

1

1 1(sign)

1 1(sign)

# SPIHT: Sorting Pass 2

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

- ◆ Is (1,4) significant? No
- ◆ Is (2,4) significant? No
- ◆ LIP = { (1,2), (2,1), (2,2), (1,4), (2,4) } LIS = { (2,1)D, (2,2)D }
- ◆ Is (2,1)D significant? No
- ◆ Is (2,2)D significant? No
- ◆ LIP = { (1,2), (2,1), (2,2), (1,4), (2,4) } LIS = { (2,1)D, (2,2)D },
- ◆ Refinement Pass 2
  - Like EZW, 1 bit for 18(1,1)

0

0

0

0

0

Bit budget = 18 bits

# SPIHT: Sorting Pass 3

- ◆  $T = 4$
- ◆ Is (1,2) significant? Yes
- ◆  $LSP = \{ (1,1), (1,3), (2,3), (1,2) \}$
- ◆ Is (2,1) significant? No
- ◆ Is (2,2) significant? Yes
- ◆  $LSP = \{ (1,1), (1,3), (2,3), (1,2), (2,2) \}$
- ◆ Is (1,4) significant? Yes
- ◆  $LSP = \{ (1,1), (1,3), (2,3), (1,2), (2,2), (1,4) \}$
- ◆ Is (2,4) significant? No
- ◆  $LIP = \{ (2,1), (2,4) \}$
- ◆ Is (2,1)D significant? No
- ◆ Is (2,2)D significant? Yes

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

1 1(sign)

0

1 0(sign)

1 1(sign)

0

0

1

# SPIHT: Sorting Pass 3

- ◆ Is (3,3) significant? Yes
- ◆ LSP = { (1,1), (1,3), (2,3), (1,2), (2,2), (1,4), (3,3) }
- ◆ Is (4,3) significant? Yes
- ◆ LSP = { (1,1), (1,3), (2,3), (1,2), (2,2), (1,4), (3,3), (4,3) }
- ◆ Is (3,4) significant? No
- ◆ LIP = { (2,1), (2,4), (3,4) }
- ◆ Is (4,4) significant? No
- ◆ LIP = { (2,1), (2,4), (3,4), (4,4) }
- ◆ LIP = { (2,1), (3,4), (3,4), (4,4) },
- ◆ LIS = { (2,1)D },
- ◆ LSP = { (1,1), (1,3), (2,3), (1,2), (2,2), (1,4), (3,3), (4,3) }

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

1 0(sign)

1 1(sign)

0

0

## ◆ Refinement Pass 3

- Like EZW, 3 bit for 18(1,1), 8(1,3), 13(2,3)

0 1 0

Bit budget = 37 bits

# Other Approaches

- ◆ Idea can be generalized to other different data structures
- ◆ For example, quad-tree
- ◆ **Sorting Pass 1**
  - 1 0 0 0 1 0 0 0
- ◆ **Refinement Pass 1**: nothing
- ◆ **Sorting Pass 2**
  - 0 0 1 0 1 1 0 0
- ◆ **Refinement Pass 2**
  - Like EZW, 1 bit for 18
- ◆ **Sorting Pass 3**
  - 1 0 1 1 0 1 1 1 0 1 1 0 0
- ◆ **Refinement Pass 3**
  - Like EZW, 3 bits for 18 8 13

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

|    |    |    |    |
|----|----|----|----|
| 0  | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

|    |    |    |    |
|----|----|----|----|
| 0  | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 0  | 0  | -6 | 4  |
| -7 | 1  | 3  | -2 |

# Scalable Coding



Trac D. Tran

ECE Department

The Johns Hopkins University

Baltimore MD 21218

# Outline

---

- ◆ Fundamentals. Main ideas. Applications
- ◆ Scalability modes
  - Quality or SNR scalability
  - Spatial scalability
  - Temporal scalability
  - Frequency scalability or data partition
  - Hybrid scalability
  - Coarse- and fine-granularity scalability
- ◆ Image scalable coding
  - Embedded zero-tree wavelet coding (EZW)
  - Set partitioning in hierarchical trees (SPIHT)
  - JPEG2000
- ◆ Video scalable coding
  - Layer coding: coarse granularity
  - Fine-granularity video coding
  - 3D sub-band video coding

# Fundamentals

---

- ◆ Scalability coding: capability of recovering physically meaningful signal information by decoding only **partial** compressed bit-stream
- ◆ Scalable coding generates a **single** coded representation (bit-stream) in a manner that facilitates the derivation of signal of many **different** resolutions and qualities at the decoder
- ◆ **Embedded** or **progressive** bit-stream: a bit stream that can be truncated at any point and the decoded signal is the same as if the signal has been originally encoded at that rate
- ◆ Embeddedness is the extreme of scalability, sometimes labeled fine-granularity scalability

# Goals and Approaches

---

- ◆ **Simulcast** coding
  - Encode the same signal several times, each with a different quality setting
  - Each of the generated bit-stream is non-scalable
  - Advantage: simple, efficient for each particular setting
  - Disadvantage: inefficient overall
- ◆ Design goal in scalable coding
  - Realizing requirement for scalability
  - Minimizing the reduction in coding efficiency
- ◆ Approach
  - Coarse-granularity scalability: only have a few layers, usually two to three only
  - Fine-granularity scalability: many layers, offer more decoding options and precise bit-rate control

# Scalability Classification

---

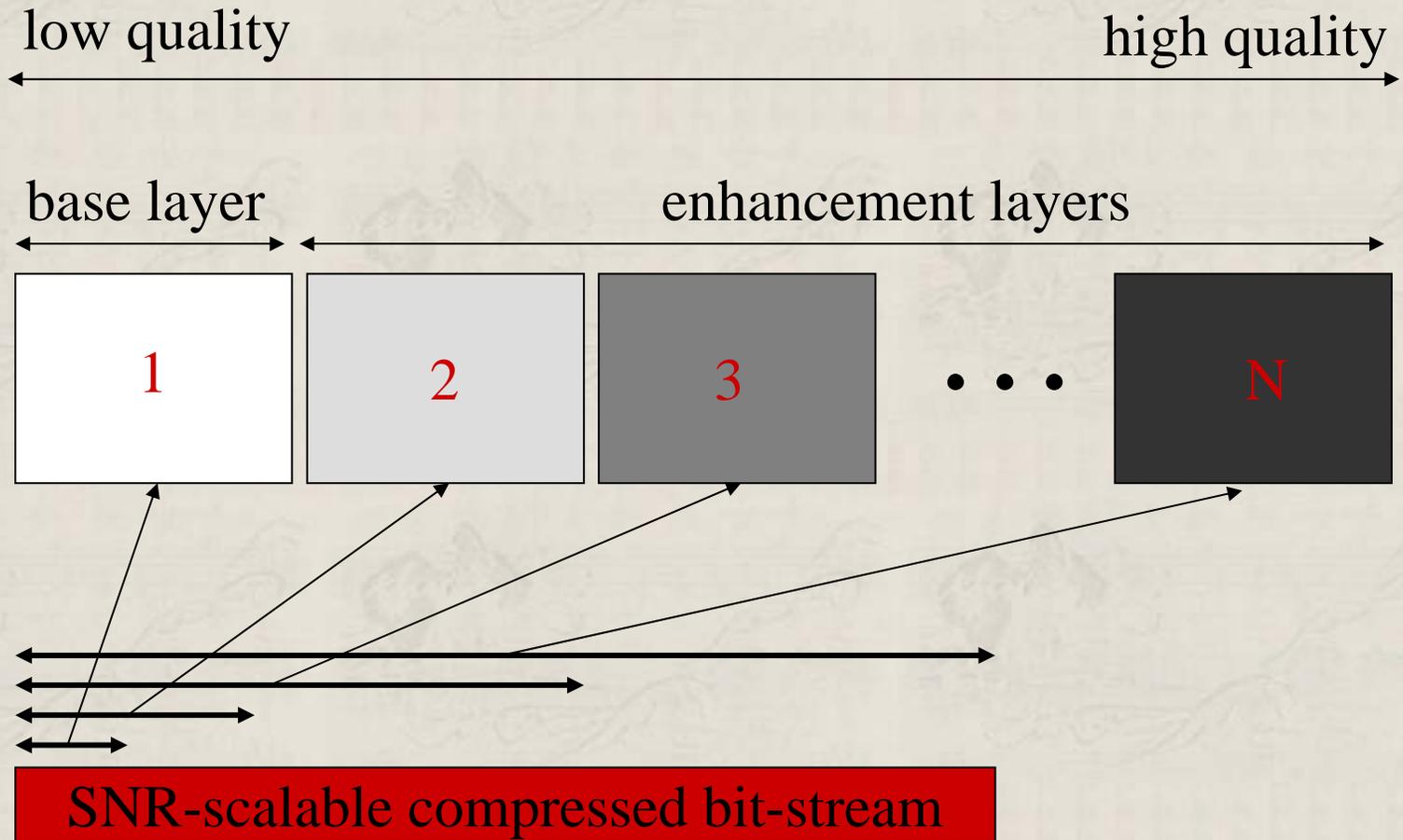
- ◆ Quality or SNR scalability
  - Represent signal with many layers, each at a **different quality level** or at different accuracy
- ◆ Spatial scalability
  - More than one layer and they can usually have **different spatial resolution**
- ◆ Temporal scalability
  - More than one layer & each can have **different temporal resolution** (frame rate)
- ◆ Frequency scalability or data partitioning
  - Single-coded bit-stream is artificially partitioned into layers, each contains **different frequency content**
- ◆ Hybrid scalability
  - **Combination** of two or more types of scalability above

# Scalable Applications

---

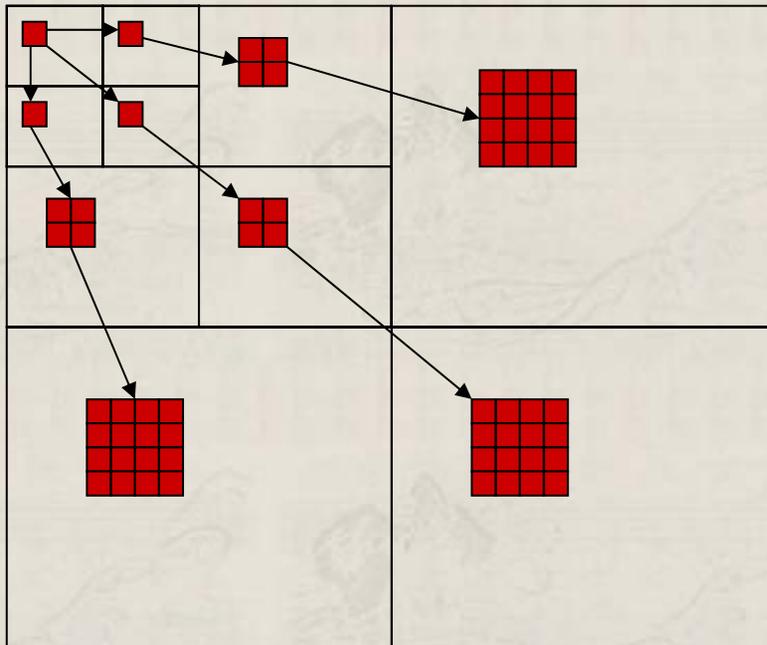
- ◆ Quality/SNR scalability
  - Digital broadcast TV or HDTV with different quality layers
  - Multi-quality video-on-demand services
  - Error-resilient video over ATM and other networks
- ◆ Spatial scalability
  - Inter-working between two different video standards
  - Layered digital TV broadcast
  - Video on LAN and computer networks
  - Error-resilient video over lossy channels
- ◆ Temporal scalability
  - Migration from low to high temporal resolution
  - Networked video. Error resilience
  - Multi-quality video-on-demand services based on decoder capability as well as communication bit-rate
- ◆ Frequency scalability
  - Error resilience

# Quality/SNR Scalability



- ◆ N layers of quality/SNR scalability

# Wavelet Zero-Tree

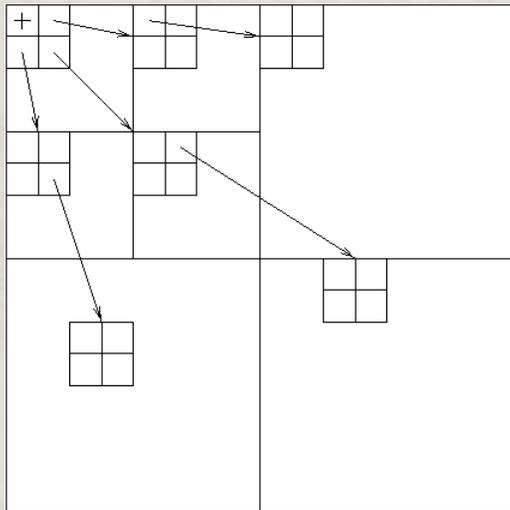


- ◆ Main observation: there is self-similarity between wavelet coefficients across different scales
- ◆ If a parent is insignificant with respect to a threshold  $T$ , i.e.  $|C| < T$ , then so are its children

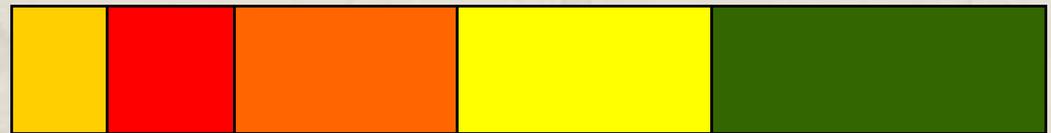
parent :  $c_{i,j}$

children :  $\{c_{2i,2j}, c_{2i+1,2j}, c_{2i,2j+1}, c_{2i+1,2j+1}\}$

# Wavelet Bit Plane Coding



Embedded bit-stream



browsing

← acceptable →

← high quality →

← lossless →

|            | sign | s               | s | s | s | s | s | s | s | s | s | s | s | s |
|------------|------|-----------------|---|---|---|---|---|---|---|---|---|---|---|---|
| <b>msb</b> | 5    | 1               | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|            | 4    | →→              |   | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|            | 3    | →→→             |   |   | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
|            | 2    | →→→→→→→         |   |   |   |   |   |   | 1 | 1 | 1 | 1 | 1 |   |
|            | 1    | →→→→→→→→→→→→→→→ |   |   |   |   |   |   |   |   |   |   |   |   |
| <b>lsb</b> | 0    | →→→→→→→→→→→→→→→ |   |   |   |   |   |   |   |   |   |   |   |   |

# EZW Coding

---

- ◆ Embedded zero-tree wavelet coding [Shapiro 1993]
  - Wavelet transform for image de-correlation
  - Exploitation of self-similarity of wavelet coefficients across different scales to predict the location of significant information
  - Further compression with adaptive arithmetic coding
- ◆ Main features
  - Bit-plane coding
  - One sorting pass and one refinement pass per bit plane with a pre-defined scan pattern
  - Use four symbols to classify wavelet coefficients
    - POS: positive significant
    - NEG: negative significant
    - ZTR: zero-tree root; parent and all children are insignificant
    - IZ: isolated insignificant; parent is insignificant but at least one of the children is significant

# Toy Example

wavelet coefficients

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

- ◆ Rank coefficients by magnitude
- ◆ Transmit coefficients bit plane by bit plane: 0 010 10011100
- ◆ Problem: how do we transmit the rank order to the decoder?

|      |    |    |   |   |   |    |    |   |   |   |   |   |    |    |   |   |
|------|----|----|---|---|---|----|----|---|---|---|---|---|----|----|---|---|
| Sign | +  | +  | + | + | + | -  | -  | + | + | + | + | + | -  | -  | + | + |
| MSB  | 1  |    |   |   |   |    |    |   |   |   |   |   |    |    |   |   |
|      | 0  | 1  | 1 |   |   |    |    |   |   |   |   |   |    |    |   |   |
|      | 0  | 1  | 0 | 1 | 1 | 1  | 1  | 1 |   |   |   |   |    |    |   |   |
|      | 1  | 0  | 0 | 1 | 1 | 1  | 0  | 0 | 1 | 1 | 1 | 1 | 1  | 1  |   |   |
| LSB  | 0  | 1  | 0 | 1 | 0 | 0  | 1  | 0 | 1 | 1 | 0 | 0 | 0  | 0  | 1 | 1 |
|      | 18 | 13 | 8 | 7 | 6 | -6 | -5 | 4 | 3 | 3 | 2 | 2 | -2 | -2 | 1 | 1 |

# Quantization & Reconstruction

|   |
|---|
| 1 |
| 0 |
| 1 |
| 1 |
| 0 |

Original  
coefficient  
 $C = 22$

|   |
|---|
| 1 |
| X |
| X |
| X |
| X |

Truncate  
4 bit planes  
Range=[16, 32)  
 $Cr = 24$

|   |
|---|
| 1 |
| 0 |
| X |
| X |
| X |

Receive 1  
refinement bit  
Range=[16, 24)  
 $Cr = 20$   
 $= 24 - 4$

|   |
|---|
| 1 |
| 0 |
| 1 |
| X |
| X |

Receive 2  
refinement bits  
Range=[20, 24)  
 $Cr = 22$   
 $= 20 + 2$

◆  $N$ -bit-plane truncation = scalar quantization with

$$\Delta = 2^N$$

# EZW Basic Algorithm

- ◆ Set initial threshold:  $T = 2^{\lfloor \log_2 |\max| \rfloor}$
- ◆ Sorting Pass – Dominant Pass
  - scan coefficients from top left corner
  - parent nodes are always scanned before children
  - For each coefficient, output a symbol among {POS, NEG, ZTR, IZ} depending on the threshold  $T$
- ◆ Refinement Pass – Subordinate Pass
  - refine the accuracy of each significant coefficient by sending one additional bit of its binary representation
- ◆ Reduce the threshold by a factor of 2:

$$T = \frac{T}{2} \quad \text{and repeat Step 2}$$

# EZW Example: First Bit Plane

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

POS =11

NEG =10

IZ =01

ZTR =00

- ◆ T=16
- ◆ Dominant Pass 1
  - POS ZTR ZTR ZTR
  - Subordinate list = {18}
- ◆ Subordinate Pass 1
  - No symbols because subordinate step  $i$  works on significant coefficients from dominant step  $i-1$  and earlier

Compressed bit-stream

11 00 00 00 – 8 bits

Reconstruction = {24 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0}

# EZW Example: 2nd Bit Plane

|    |    |    |    |
|----|----|----|----|
| *  | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

POS =11

NEG =10

IZ =01

ZTR =00

- ◆ T=8
- ◆ Dominant Pass 2

- ZTR IZ ZTR POS POS IZ IZ
- Subordinate list = {18 8 13}

Compressed bit-stream

00 01 00 11 11 01 01 – 14 bits

- ◆ Subordinate Pass 2
- Send the bit plane of coefficients involved in Dominant Pass 1

0 – 1 bit

Reconstruction = {20 12 12 0 0 0 0 0 0 0 0 0 0 0 0}

Bit budget = 23 bits

# EZW Example: 3rd Bit Plane

|    |    |    |    |
|----|----|----|----|
| *  | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| *  | *  | -6 | 4  |
| -7 | 1  | 3  | -2 |

POS = 11  
 NEG = 10  
 IZ = 01  
 ZTR = 00

◆ T=4

◆ Dominant Pass 3

Compressed bit-stream

- ZTR POS NEG NEG IZ NEG POS IZ IZ
- Subordinate list = {18,8,13,6,-5,-7,-6,4}

◆ Subordinate Pass 3

00 11 10 10 01 10 11 01 01 – 18 bits

- Send the bit plane of coefficients involved in Dominant Pass 2

001 – 3 bits

Reconstruction = {18 10 14 6 -6 -6 -6 6 0 0 0 0 0 0 0}

Bit budget = 44 bits

# EZW Decoding

---

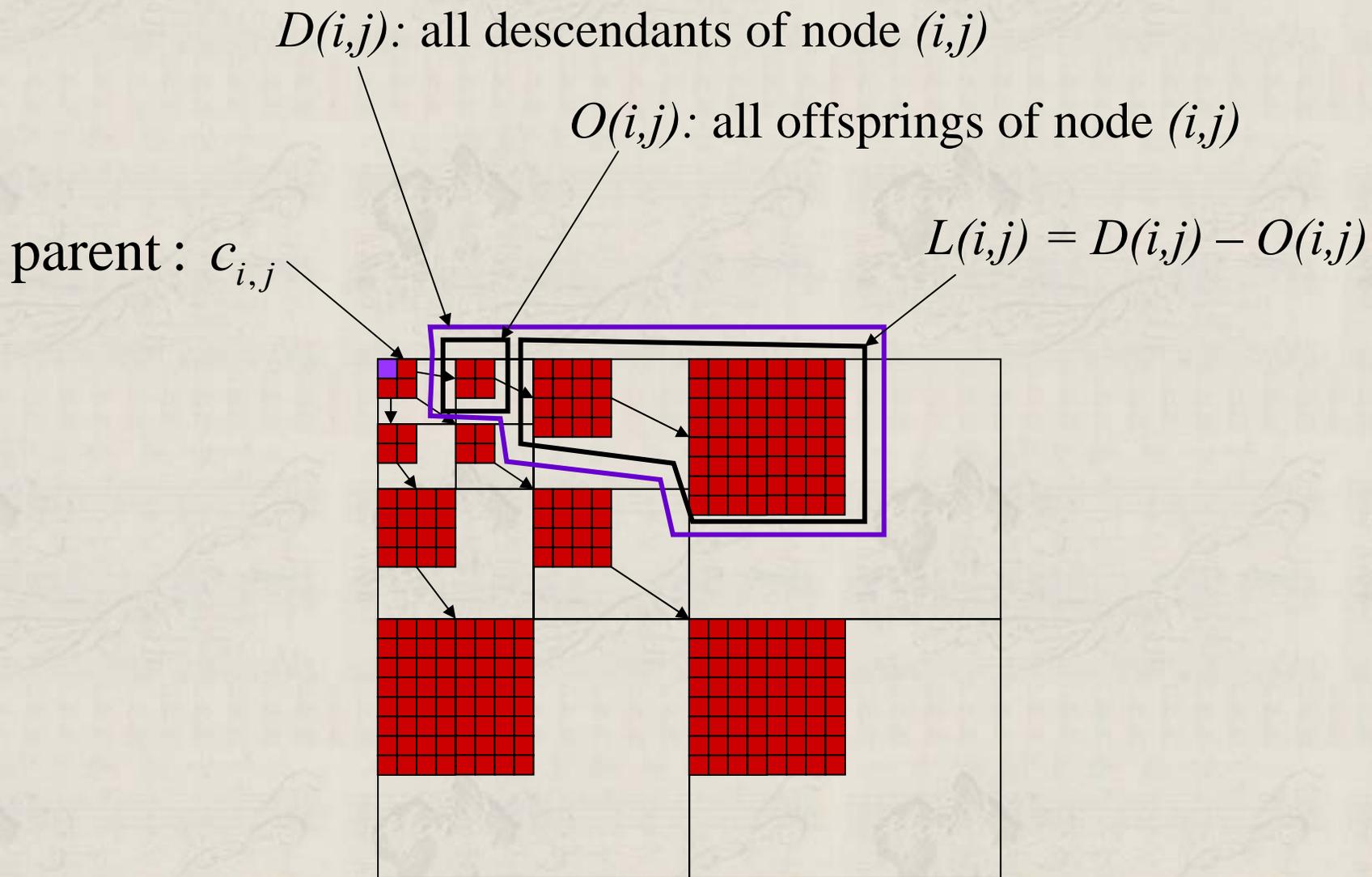
- ◆ The decoder needs
  - Initial threshold  $T$  (or the max absolute value of all coefficients)
  - Original image size
  - Number of wavelet decomposition levels
  - Encoded bit-stream
- ◆ Decoding process
  - Decode the arithmetic-encoded bit-stream into a stream of symbols
  - Based on the side information, create data structures of appropriate sizes
  - Traverse the encoding algorithm

# SPIHT

---

- ◆ Most popular extension of EZW [Said-Pearlman 1996]
- ◆ Improves EZW by having more efficient **significance map** coding based on sophisticated set partitioning algorithm
- ◆ SPIHT has 3 lists
  - **LIP**: list of insignificant pixels (individual insignificant coefficients)
  - **LIS**: list of insignificant lists (insignificant trees)
  - **LSP**: list of significant pixels (significant coefficients)
- ◆ SPIHT defines 2 types of trees
  - **Type D**: check all descendants for significance
  - **Type L**: check all descendants except immediate children
- ◆ Other features
  - Root node is checked independently of the rest of the tree
  - SPIHT sorting pass checks significance of LIP & LIS elements, then moves significant coefficients to LSP

# SPIHT Zero-Tree



# Set Partitioning Rules

---

- ◆ Initial partition is formed with the set  $\{(i,j)\}$  and  $D(i,j)$  for all coefficients  $(i,j)$  in the lowpass subband
- ◆ If  $D(i,j)$  is significant, it is partitioned into  $L(i,j)$  plus four single-element sets in  $O(i,j)$
- ◆ If  $L(i,j)$  is significant, then it is partitioned into 4 sets  $D(k,l)$  where

$$(k,l) \in O(i,j)$$

# SPIHT Basic Algorithm

- ◆ Initialization. Compute initial threshold. LIP: all root nodes (in lowpass subband). LIS: all trees (type D). LSP: empty
- ◆ Check significance of all coefficients in LIP
  - If significant, output 1 followed by a sign bit & move it to LSP
  - If insignificant, output 0
- ◆ Check significance of all trees in LIS
  - For type-D tree
    - If significant, output 1 & proceed to code its children
      - If a child is significant, output 1, sign bit, & add it to LSP
      - If a child is insignificant, output 0 and add it to the end of LIP
      - If the child has descendants, move the tree to the end of LIS as type L, otherwise remove it from LIS
    - If insignificant, output 0
  - For type-L tree
    - If significant, output 1, add each of the children to the end of LIS as type D and remove the parent tree from LIS
    - If insignificant, output 0
- ◆ Refinement pass, like EZW
- ◆ Decrease the threshold by a factor of 2. Go to Step 2.

# SPIHT Example: First Pass

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

Compressed bit-stream

## ◆ Initialization

- $T=16$
- $LIP=\{(1,1)\}$ .  $LIS=\{(1,1)D\}$ .  $LSP=\{\}$

## ◆ Dominant Pass 1

- $(1,1)$  significant? Yes
- $LSP=\{(1,1)\}$
- $(1,1)D$  significant? No

1 1(sign)

0

## ◆ Subordinate Pass 1

- No symbols, like EZW

$LIP=\{\}$ .  $LIS=\{(1,1)D\}$ .  $LSP=\{(1,1)\}$

Bit budget = 3 bits

# SPIHT: Sorting Pass 2

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

- ◆ T=8
- ◆ (1,1)D significant? Yes
- ◆ (1,2) significant? No
- ◆ (2,1) significant? No
- ◆ (2,2) significant? No
- ◆ LIP = { (1,2), (2,1), (2,2) }. LIS = { (1,1)L }
- ◆ (1,1)L significant? Yes
- ◆ LIS = { (1,2)D, (2,1)D, (2,2)D }
- ◆ Is (1,2)D significant? Yes
- ◆ Is (1,3) significant? Yes
- ◆ LSP = { (1,1), (1,3) }
- ◆ Is (2,3) significant? Yes
- ◆ LSP = { (1,1), (1,3), (2,3) }

1

0

0

0

1

1

1 1(sign)

1 1(sign)

# SPIHT: Sorting Pass 2

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

- ◆ Is (1,4) significant? No
- ◆ Is (2,4) significant? No
- ◆ LIP = { (1,2), (2,1), (2,2), (1,4), (2,4) } LIS = { (2,1)D, (2,2)D }
- ◆ Is (2,1)D significant? No
- ◆ Is (2,2)D significant? No
- ◆ LIP = { (1,2), (2,1), (2,2), (1,4), (2,4) } LIS = { (2,1)D, (2,2)D },

## ◆ Refinement Pass 2

- Like EZW, 1 bit for 18(1,1)

0

0

0

0

0

Bit budget = 18 bits

# SPIHT: Sorting Pass 3

- ◆  $T = 4$
- ◆ Is (1,2) significant? Yes
- ◆  $LSP = \{ (1,1), (1,3), (2,3), (1,2) \}$
- ◆ Is (2,1) significant? No
- ◆ Is (2,2) significant? Yes
- ◆  $LSP = \{ (1,1), (1,3), (2,3), (1,2), (2,2) \}$
- ◆ Is (1,4) significant? Yes
- ◆  $LSP = \{ (1,1), (1,3), (2,3), (1,2), (2,2), (1,4) \}$
- ◆ Is (2,4) significant? No
- ◆  $LIP = \{ (2,1), (2,4) \}$
- ◆ Is (2,1)D significant? No
- ◆ Is (2,2)D significant? Yes

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

1 1(sign)

0

1 0(sign)

1 1(sign)

0

0

1

# SPIHT: Sorting Pass 3

- ◆ Is (3,3) significant? Yes
- ◆ LSP = { (1,1), (1,3), (2,3), (1,2), (2,2), (1,4), (3,3) }
- ◆ Is (4,3) significant? Yes
- ◆ LSP = { (1,1), (1,3), (2,3), (1,2), (2,2), (1,4), (3,3), (4,3) }
- ◆ Is (3,4) significant? No
- ◆ LIP = { (2,1), (2,4), (3,4) }
- ◆ Is (4,4) significant? No
- ◆ LIP = { (2,1), (2,4), (3,4), (4,4) }
- ◆ LIP = { (2,1), (3,4), (3,4), (4,4) },
- ◆ LIS = { (2,1)D },
- ◆ LSP = { (1,1), (1,3), (2,3), (1,2), (2,2), (1,4), (3,3), (4,3) }

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

1 0(sign)

1 1(sign)

0

0

## ◆ Refinement Pass 3

- Like EZW, 3 bit for 18(1,1), 8(1,3), 13(2,3)

0 1 0

Bit budget = 37 bits

# Other Approaches

- ◆ Idea can be generalized to other different data structures
- ◆ For example, quad-tree
- ◆ **Sorting Pass 1**
  - 1 0 0 0 1 0 0 0
- ◆ **Refinement Pass 1**: nothing
- ◆ **Sorting Pass 2**
  - 0 0 1 0 1 1 0 0
- ◆ **Refinement Pass 2**
  - Like EZW, 1 bit for 18
- ◆ **Sorting Pass 3**
  - 1 0 1 1 0 1 1 1 0 1 1 0 0
- ◆ **Refinement Pass 3**
  - Like EZW, 3 bits for 18 8 13

|    |    |    |    |
|----|----|----|----|
| 18 | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

|    |    |    |    |
|----|----|----|----|
| 0  | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 8  | 13 | -6 | 4  |
| -7 | 1  | 3  | -2 |

|    |    |    |    |
|----|----|----|----|
| 0  | 3  | 2  | 2  |
| 6  | -5 | 1  | -2 |
| 0  | 0  | -6 | 4  |
| -7 | 1  | 3  | -2 |

# JPEG2000 Image Coding

---

- ◆ About JPEG2000 (ISO/IEC15444)
- ◆ Objectives of JPEG2000
  - ◆ To provide new functionalities and features that current standards fail to support
  - ◆ To support advanced applications in the new millennium
  - ◆ To extend the applicability of image coding in more applications
  - ◆ To allow imaging applications to be interactive and adaptive

# JPEG2000 vs. JPEG

## ◆ Key Advantages

- ◆ Wavelet based – better rate-distortion performance
- ◆ Scalable by resolution, quality, color channel, location in image
- ◆ Lossless encoding, including lossy to lossless scalability
- ◆ Error resilience
- ◆ Region-of-Interest coding and progressive decoding

Compression ratio: 100:1



# JPEG2000 Flexible Decoding

Encoder choices:

tiling,  
lossy/lossless  
+ other choices



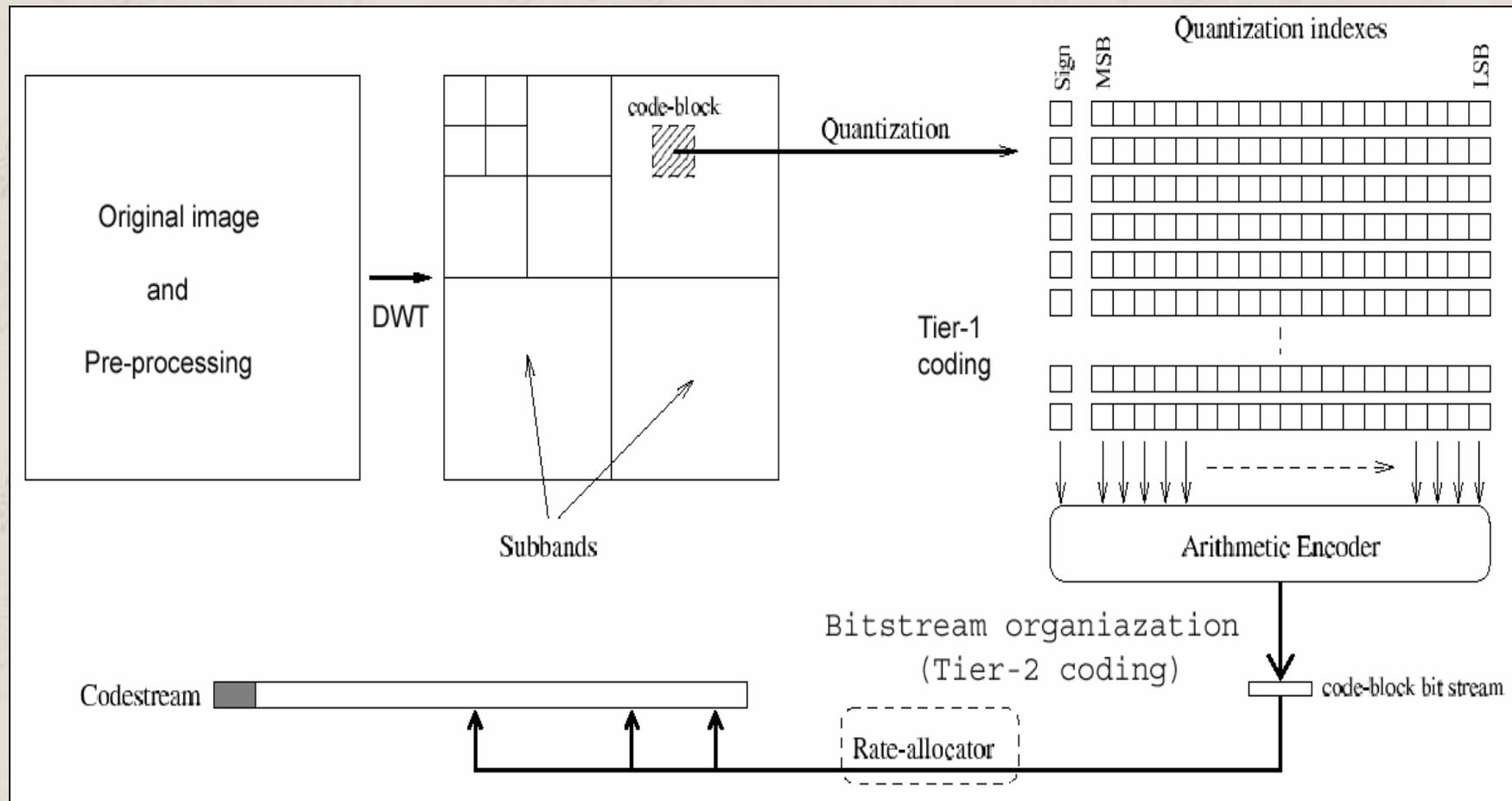
Decoder choices:

image resolution,  
image fidelity,  
region-of-interest,  
Fixed-rate,  
components

JPEG 2000 offers flexible decoding



# JPEG2000 Compression Scheme



# Part 1: Discrete Wavelet Transform

---

- ◆ Inherent to normal DWT:
  - Multi-resolution image representation
  - Eliminate blocking artifacts at high compression ratio
  - Each subband can be quantized differently
- ◆ Special techniques:
  - Provide integer filter (e.g. (5,3) filter) to support lossless and lossy compression within a single compressed bit-stream;
  - Line-based DWT and lifting implementations to reduce the memory requirement and computational complexity.

Except for a few special case, e.g., the (5,3) integer filter, the DWT is generally more computationally complexity ( $\sim 2$  to  $3$ ) than the block-based DCT; and DWT also requires more memory than DCT.

# Line-based DWT Implementation



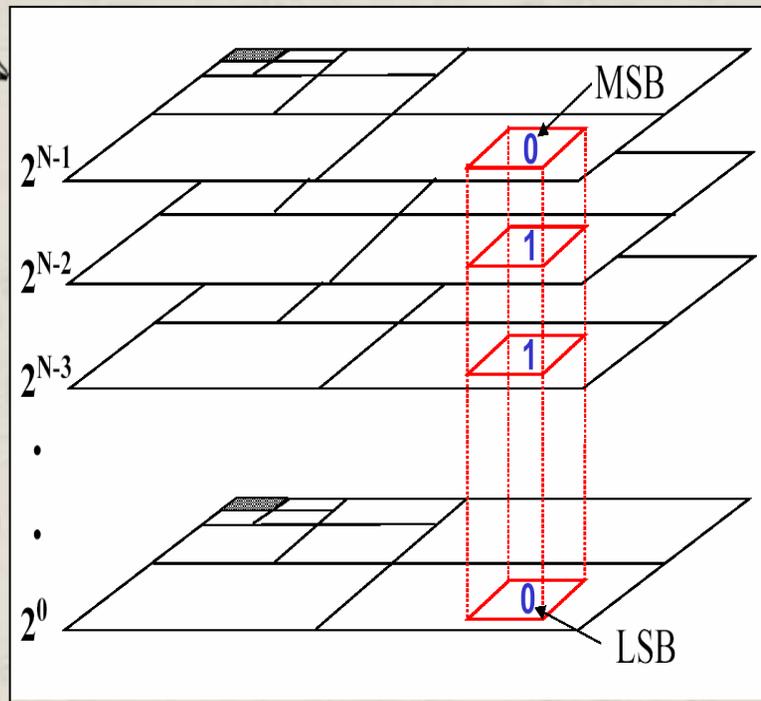
- ◆ There is no need to buffer an entire image in order to perform wavelet transform.
- ◆ Depending on filter lengths and decomposition levels, a line of wavelet coefficients can be made available only after processing a few lines of the input image.

# Part 2: Quantization

- ◆ Embedded Quantization:

Quantization index is encoded bit by bit, starting from Most Significant Bit (MSB) to Least Significant Bit (LSB).

- ◆ Example:



Wavelet coefficient = 209

Quantizer step size  $\Delta_b = 2$

Quantization index =  $\lfloor 209/2 \rfloor = 104$   
= 01101000;

Dequantized value based on fully decoded index:  
 $(104+0.5)*2 = 209$ ;

Decoding value after decoding 3 bit planes:

- Decoded index = 011 = 3;

- Step size =  $2*32=64$

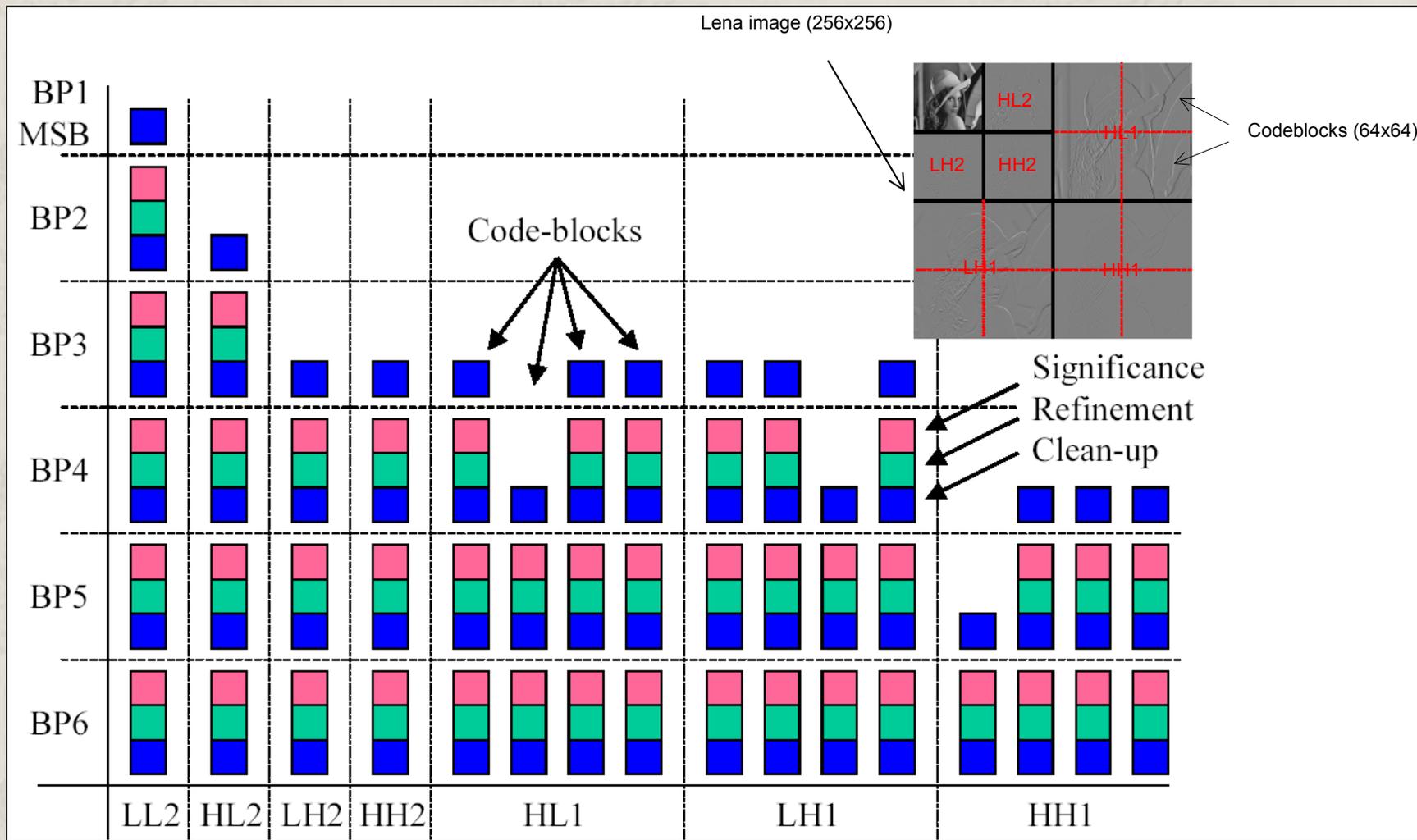
- Dequantized value =  $(3+0.5)*64 = 224$

# Part 3: Entropy Coding (Tier-1 )

---

- ◆ Tier-1 Entropy coding
  - Each bit-plane is individually coded by the context-based adaptive binary arithmetic coding (JBIG2 MQ-coder)
  - Each bit plane is partitioned into blocks, named *code-blocks*, which are encoded independently
  - Each bit plane of each block is encoded in three *sub-bit-plane passes*
    - Significance propagation pass
    - Magnitude refinement pass
    - Clean-up pass

# Example of Bit-plane Coding

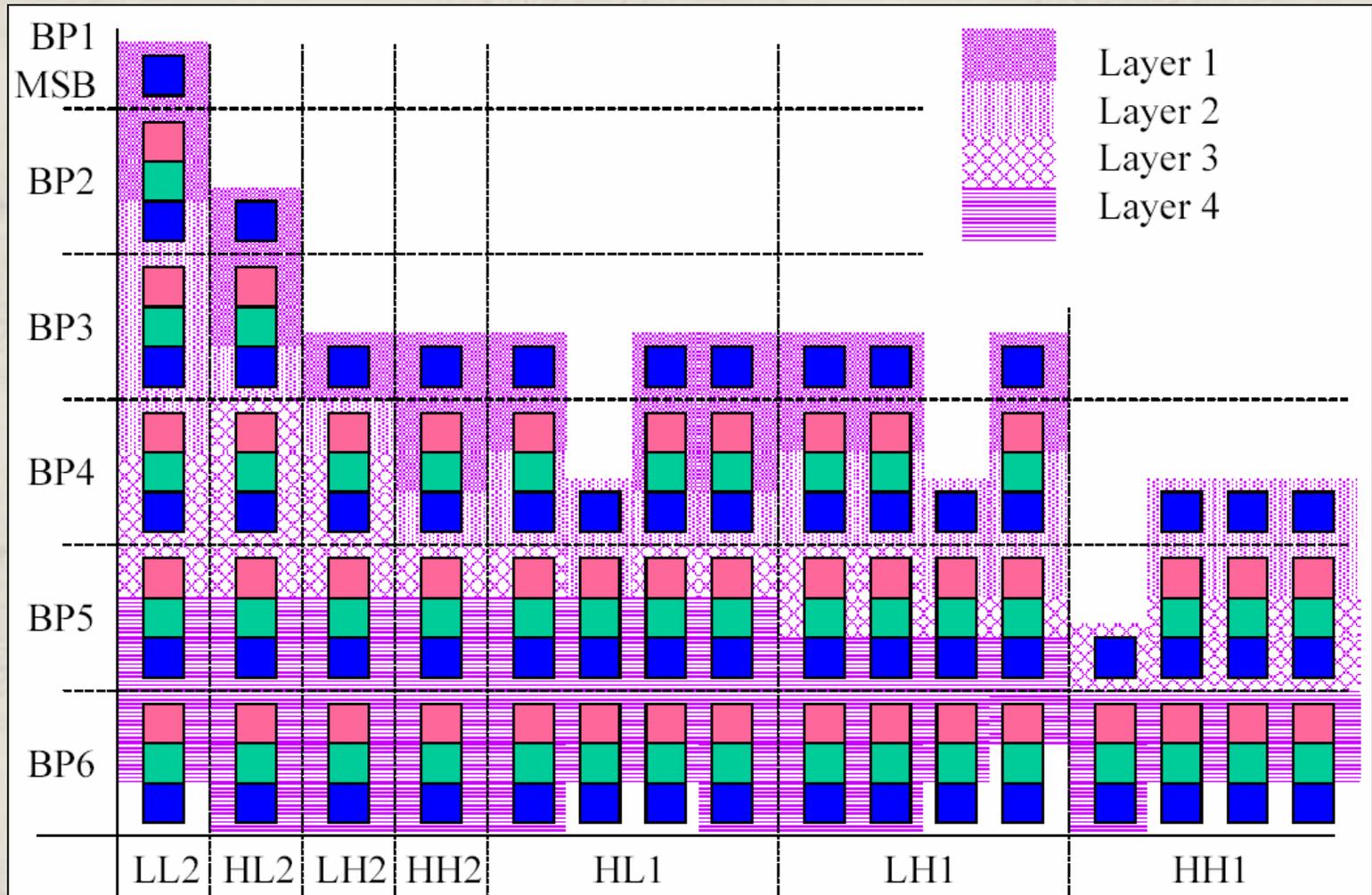


# Part 4: Bit stream Organization (Tier 2)

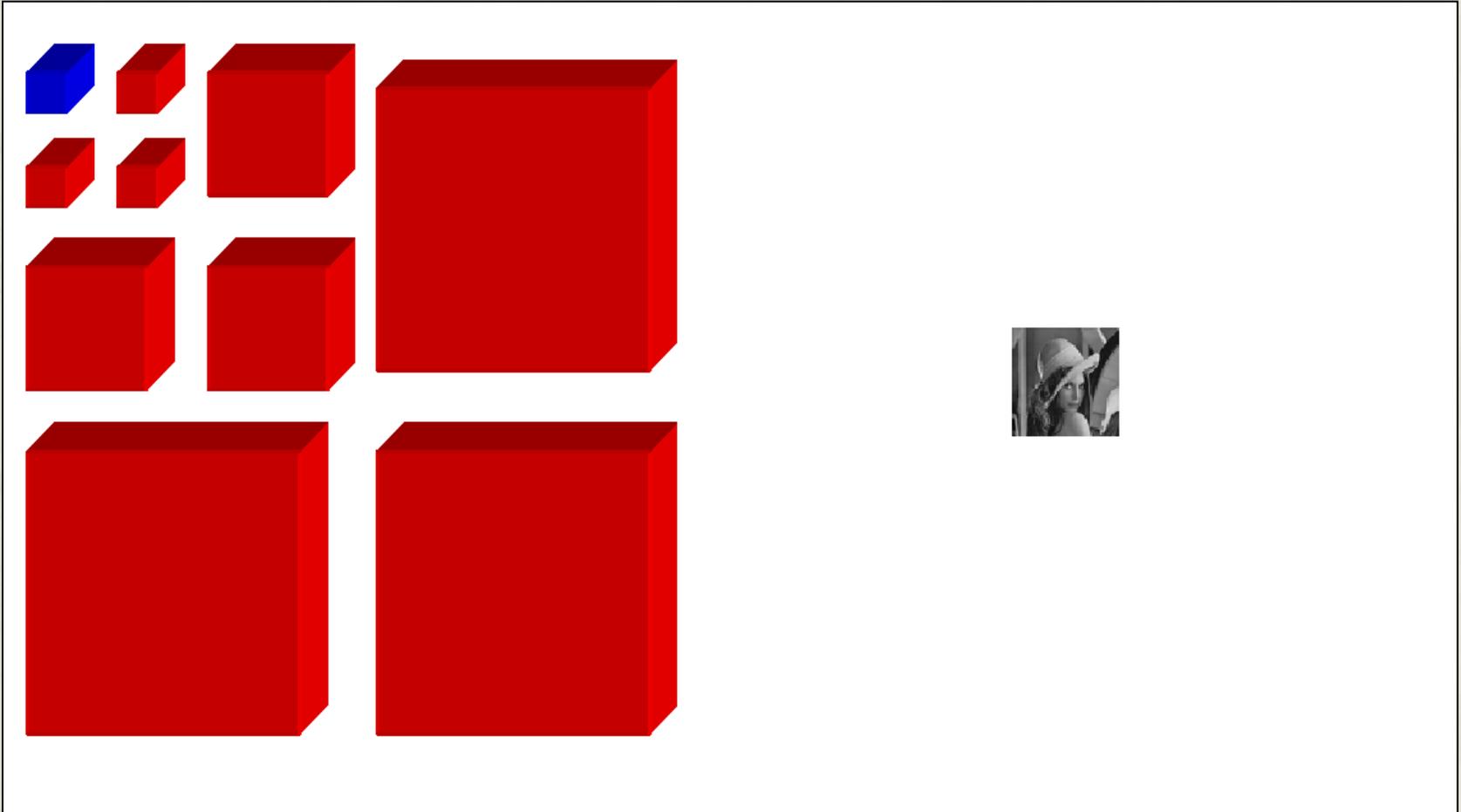
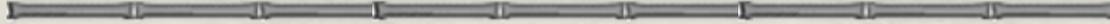
---

- ◆ Tier-1 generates a collection of bitstreams
  - One independent bitstream from each code block
  - Each bitstream is embedded
- ◆ Tier-2 multiplexes the bitstreams for inclusion in the codestream and signals the ordering of the resulting coded bitplane passes in an efficient manner.
- ◆ Tier-2 coded data can be rather easily parsed
- ◆ Tier-2 enables SNR, resolution, spatial, ROI and arbitrary progression and scalability

# Example: Bit-stream Organization



# Example: Progressive Resolution





# JPEG2000 Summary

---

- ◆ JPEG2000 offers the state-of-the-art features
  - Superior low bit rate performance and coding efficiency (up to 30% compared with DCT)
  - Lossless and lossy compression
  - Progressive transmission by pixel accuracy and resolution
  - Region-of-Interest coding
  - Random codestream access and processing
  - Error resilience
  - Open architecture
  - Content-based description
  - Side channel spatial information (transparency)
  - Protective image security
  - Continuous-tone and bi-level compression