CS 6351 DATA COMPRESSION

THIS LECTURE:
TRANSFORMS PART I

Instructor: Abdou Youssef

OBJECTIVES OF THIS LECTURE

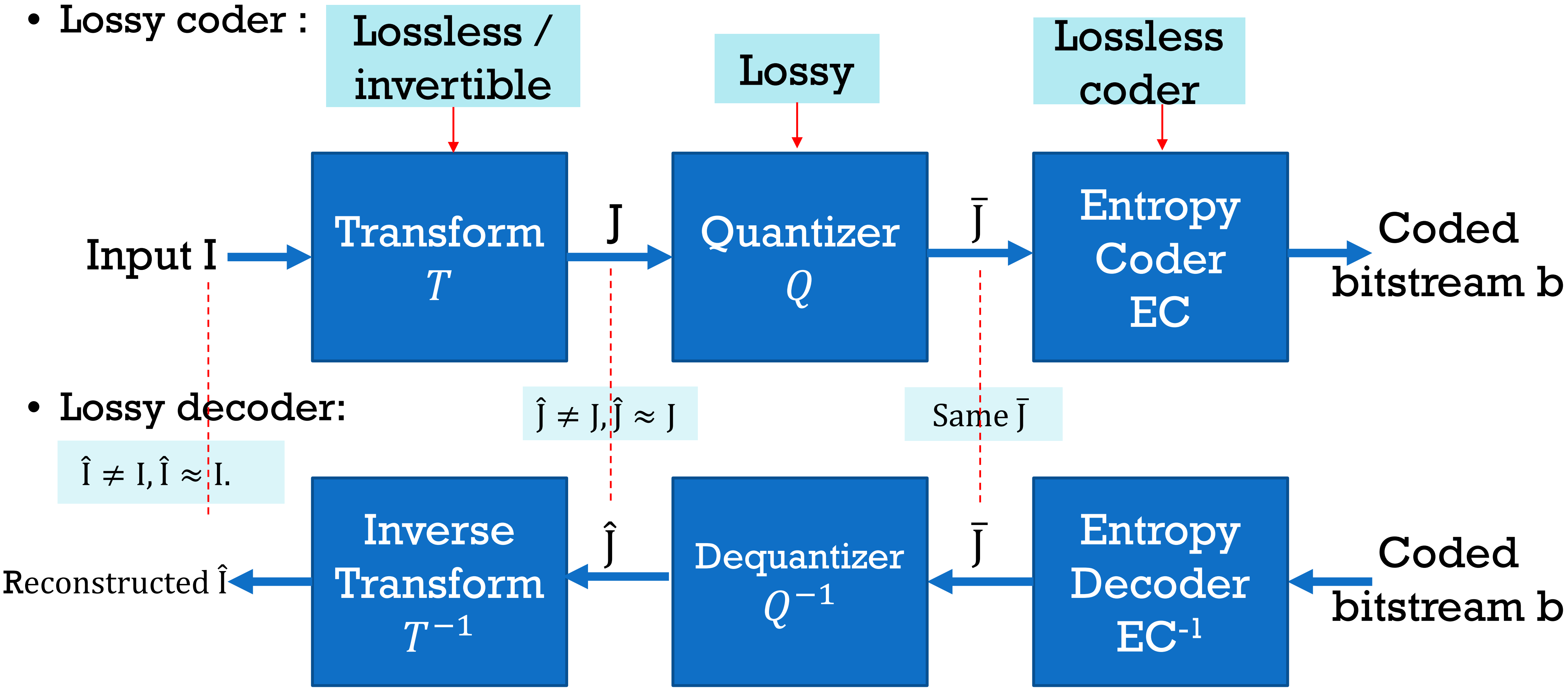
By the end of this lecture, you will be able to:

- Explain linear transforms, as a matrix-vector multiplication
- Describe the matrices that specify popular transforms
- Apply popular transforms to 1D input and 2D input
- Derive the time complexity of 1D and 2D transforms, and translate that into real time for real-world situations
- Appreciate the need for speeding up transforms
- Utilize complex numbers, especially as used in the definition of the Discrete Fourier Transform

OUTLINE

- Review of the lossy compression framework, and the role of each of its 3 components
- Review of the desirable properties of transforms for data compression
- Definition of linear transforms in general, both for 1D and 2D input signals
- Algorithm for transforms, and its time complexity
- The matrices that specify each of the transforms to be focused on
- Review of complex numbers
- Revisiting the Discrete Fourier Transform (defined in terms of complex numbers)

GENERAL SCHEME OF LOSSY COMPRESSION



The entire data loss in limited to the quantizer

LOSSY COMPRESSION
-- MOTIVATION --

1. We saw that lossless compression does not yield high enough compression ratios (or low enough bitrates) needed for demanding applications like in images and videos, especially in increasingly high definition
2. Also, lossless compression cannot compress every input stream, while we want to have control in imposing any target bitrate at will or as needed
3. Lossless compression has no way of exploiting the characteristics/limitations of the human audio-visual systems

- The top two reasons force us to look for alternatives to lossless compression
- The 3rd reason opens the door for lossy compression which, among other things, affords us the ability to control the bitrate at will (in tradeoff with quality)

QUANTIZATION REVIEW

- Quantization operates on numerical data
- Quantization reduces the precision of the data to save on bits
- Scalar quantization quantizes each number in the input <u>individually</u> and <u>independently</u> of other (nearby/possibly-correlated) input numbers
- Optimal non-uniform quantizers do take advantage of data distributions (e.g., smaller more numerous intervals in denser regions), and thus indirectly exploit redundancy for reducing error of reconstruction while achieving a certain bitrate, but:
  - They pay no attention to correlations w.r.t. preserving audio-visual patterns
- Therefore, if applied directly to audio-visual data, they incur considerable audio-visual error/loss/distortion

TRANFORMS (1/2)

- To prevent the damage of quantization while taking advantage of its benefits (reduced bitrate), we need:
  - a "pre-processing" stage, which we call transforms
- Specifically, the transforms must have the following properties:
  - **Decorrelation of data**
  - **Separation of data** into vision-sensitive data and vision-insensitive data
  - **Energy compaction**: concentrating the important data into a very small subset
  - **Invertability:** since data loss should occur only in quantization, transforms must be lossless
- Advantage of decorrelation of the data
  - Quantizing decorrelated data causes less blurring of contrasts (e.g., edges) than quantizing correlated data
  - In general, quantizing decorrelated data causes less loss of visual/audio patterns than quantizing correlated data

TRANFORMS (2/2)

- Advantages of separation of data into vision-sensitive (i.e., important data) and vision-insensitive (i.e., unimportant data):
  - We can quantize more the less important data (thus achieving a lot of compression)
    - By having a separate quantizer with fewer, larger intervals
  - We can quantize lightly the more important data (thus retaining quality/fidelity)
    - By having a separate quantizer with more, smaller intervals
    - With energy compaction, only a small subset of the data need to be quantized lightly, which enables us to preserve quality while still compressing a lot
  - Vision-sensitivity is a continuum, i.e., the level of importance of data is not just binary but changes progressively
  - We can progressively adjust the harshness (i.e., level) of the quantization based on the level of importance of the data
    - By having a separate quantizer for each level of importance, where the number of intervals is adapted to the level of importance

FOCUS OF THIS TOPIC

- We will focus on designing and studying transforms that have those aforementioned desirable properties
- It so happens that many such transforms already exist and were initially designed for other purposes, such as the Fourier Transform, the Haar Transform, the Hadamard Transform, and the Walsh transform
- In addition, some transforms were designed with data compression in mind, most notably the Discrete Cosine Transform (DCT)
- Therefore, we will focus on
  - The Discrete Fourier Transform (DFT)
  - The Discrete Cosine Transform (DCT)
  - The Hadamard Transform
  - The Walsh Transform
  - The Haar Transform

- There is one more worthy transform: the Karhunen-Loeve (KL) Transform
- Optimal w.r.t. data decorrelation and energy compaction
- But it is data-dependent (the transform matrix is different for different inputs)
- Therefore, it is slow and unsuitable for compression b/c the transform matrix has to be computed and stored every time

DIFFERENT PERSPECTIVES OF TRANSFORMS

- To effectively utilize and fully understand the workings and effects of transforms, we will study them from several different perspectives
  - The computational, end-user perspective
  - The vector space perspective
  - The frequency perspective
  - The statistical perspective

BUT FIRST: REVIEW OF MATRICES (1/4)

- **Definition**: A ***matrix*** A is a table of numbers (or a 2D array in programming). The lines of the table are called ***rows***, and the columns retain their name as ***columns***. If the matrix A has n rows and m columns, we say that the matrix A is an $n \times m$ matrix. n and m are called ***dimensions*** of A.
- Indexing notation of matrices:
  - The rows are labeled from 1 to n top to bottom (in math and Matlab), but sometimes from 0 to $n-1$
  - The columns are labeled 1 to m left to right (in math and Matlab), but sometimes from 0 to $m-1$
  - We denote the entry (number) located at the intersection of row i and column j as a_{ij}
  - We denote $A = \left(a_{ij}\right)_{\substack{1\le i\le n\\ 1\le j\le m}}$ or simply $A = (a_{ij})$
- **Definition**: A matrix is called a ***square matrix*** if $n = m$

Example of a matrix:

row 2

$$A = \begin{bmatrix} 1 & 0 & 5 & 3 \\ 2 & 3 & 1 & 4 \\ 8 & -1 & 1 & 2 \\ 6 & 3 & 4 & 8 \\ 5 & 2 & 8 & 9 \end{bmatrix}$$

a_{42}

column 4

MATRIX OPERATIONS (2/4)
-- MATRIX ADDITION AND SUBTRACTION --

- We can add and subtract two matrices if they have the same dimensions $n \times m$
- The addition and subtraction are defined component-wise, so the sum of two $n \times m$ matrices $A = (a_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$ and $B = (b_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$ is a matrix $C = A + B$, $C = (c_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$, where $c_{ij} = a_{ij} + b_{ij}$
- Similarly, $A - B$ is a new matrix $D = (d_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$, where $d_{ij} = a_{ij} - b_{ij}$
- Example:
  - Let $A = \begin{bmatrix} 1 & 5 \\ 3 & 4 \\ 2 & 1 \end{bmatrix}$ and $B = \begin{bmatrix} 6 & 2 \\ 3 & 1 \\ 4 & 2 \end{bmatrix}$,
  - Then $C = A + B = \begin{bmatrix} 1 & 5 \\ 3 & 4 \\ 2 & 1 \end{bmatrix} + \begin{bmatrix} 6 & 2 \\ 3 & 1 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} 7 & 7 \\ 6 & 5 \\ 6 & 3 \end{bmatrix}$ and $D = A - B = \begin{bmatrix} 1 & 5 \\ 3 & 4 \\ 2 & 1 \end{bmatrix} - \begin{bmatrix} 6 & 2 \\ 3 & 1 \\ 4 & 2 \end{bmatrix} = \begin{bmatrix} -5 & 3 \\ 0 & 3 \\ -2 & -1 \end{bmatrix}$

Observe that matrix addition is commutative and associative:
$\forall\ n \times m$ matrices A, B and C, we have
- $A + B = B + A$
- $(A + B) + C = A + (B + C)$

MATRIX OPERATIONS (3/4)

-- MATRIX MULTIPLICATION --

- Multiplication of a matrix by a number: Let $A = (a_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$ be a matrix and x is a number, then

  $xA = (b_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$ where $b_{ij} = x \times a_{ij} = xa_{ij}$

Example:

$$3\begin{bmatrix}1 & 5\\3 & 4\\2 & 1\end{bmatrix} = \begin{bmatrix}3 & 15\\9 & 12\\6 & 3\end{bmatrix}, -\begin{bmatrix}1 & 5\\3 & 4\\2 & 1\end{bmatrix} = \begin{bmatrix}-1 & -5\\-3 & -4\\-2 & -1\end{bmatrix}$$

- Multiplication of two matrices: You can multiply a matrix A with a matrix B iff the number of columns of A is equal to the number of rows of B

$$\begin{bmatrix}1 & 5\\3 & 4\\2 & 1\end{bmatrix} \cdot \begin{bmatrix}2 & 4 & 1 & 3\\1 & 2 & 4 & 5\end{bmatrix} = \begin{bmatrix}7 & 14 & 21 & 28\\10 & 20 & 19 & 29\\5 & 10 & 6 & 11\end{bmatrix}$$

- **Matrix multiplication** :

  Let $A = (a_{ij})_{\substack{1\le i\le n\\1\le j\le m}}$ be an $n \times m$ matrix, and $B = (b_{ij})_{\substack{1\le i\le m\\1\le j\le p}}$ an $m \times p$ matrix.

  Then the product AB is an $n \times p$ matrix $C = (c_{ij})_{\substack{1\le i\le n\\1\le j\le p}}$, where $c_{ij} = \sum_{k=1}^{m} a_{ik}b_{kj}$.

MATRIX OPERATIONS (4/4)
-- MATRIX MULTIPLICATION EXAMPLE --

- Example of matrix multiplication

- Example: $\begin{bmatrix} 1 & 5 \\ 3 & 4 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 & 1 & 3 \\ 1 & 2 & 4 & 5 \end{bmatrix} = \begin{bmatrix} 7 & 14 & 21 & 28 \\ 10 & 20 & 19 & 29 \\ 5 & 10 & 6 & 11 \end{bmatrix}$

$$\begin{bmatrix} 1 & 5 \\ \boxed{3 \;\; 4} \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 4 & \boxed{\begin{matrix} 1 \\ 4 \end{matrix}} & 3 \\ 1 & 2 & & 5 \end{bmatrix} = \begin{bmatrix} 7 & 14 & 21 & 28 \\ 10 & 20 & 19 & 29 \\ 5 & 10 & 6 & 11 \end{bmatrix}$$

Since 19 is in row 2 and column 3, we compute it by doing an inner-product of row 2 of matrix A with column 3 of matrix B to get the entry: $3 \times 1 + 4 \times 4 = 3 + 16 = 19$

THE COMPUTATIONAL END-USER PERSPECTIVE
-- MATRIX FORMULATION OF TRANSFORMS --

- Every transform studied here is specified by a square $N \times N$ matrix A
- The size N of the matrix is equal to the length of the input data
- **Definition**: A transform specified by an $N \times N$ matrix A is a mapping that maps any input column vector x (of components x: $x_0, x_1, \ldots, x_{N-1}$) into another column vector $\boldsymbol{y = Ax}$. That is, a transform is a *matrix multiplication* of the transform-matrix A and the input signal x
- Visually, assuming $m = N - 1$ and the indexing starts from 0, $y = Ax$ is

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & \ldots & a_{0m} \\ a_{10} & a_{11} & \ldots & a_{1m} \\ a_{20} & a_{21} & \ldots & a_{2m} \\ & \ddots & & \\ a_{m0} & a_{m1} & \ldots & a_{mm} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_m \end{bmatrix}$$

THE COMPUTATIONAL END-USER PERSPECTIVE
-- INVERSE TRANSFORM --

- To ensure invertability of the transform, the matrix A is non-singular, i.e., has an inverse matrix A^{-1}, where $AA^{-1} = A^{-1}A = I_N$ ($N \times N$ identity matrix)
- **Inverse transform**: it maps y to x: $\boldsymbol{x = A^{-1}y}$, i.e., to get x back, simply multiply y by matrix A^{-1}

$$I_N = \begin{bmatrix} 1 & & \cdots & 0 \\ 0 & 1 & & \\ \vdots & & \ddots & \vdots \\ 0 & & \cdots & 1 \end{bmatrix}$$

- Notes:
  - When the input data stream is x: $x_0, x_1, \ldots, x_{N-1}$, we express it as a column vector:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

The transpose of a matrix B, denoted B^T, is derived from B by turning every row of B into a column and every column into a row, so that $B^T_{ij} = B_{ji}$

  - To save on writing space, we often write x in its transpose form (i.e., as a row) as:

$$x^T = (x_0, x_1, x_2, \ldots, x_{N-1})$$

ALGORITHMICS OF TRANSFORMS

- For input $x^T = (x_0, x_1, x_2, \dots, x_{N-1}), A_N = (a_{kl}), 0 \le k \le N-1, 0 \le l \le N-1,\; y = A_N x$
- Let $y^T = (y_0, y_1, y_2, \dots, y_{N-1})$
- We have for each k:
  - $y_k = a_{k0}x_0 + a_{k1}x_1 + a_{k2}x_2 + \cdots a_{k,N-1}x_{N-1}$
  - That is, to get y_k, multiply (inner-product) row k of A_N with column x

```
Procedure transform(input: x[0:N − 1], A[0:N − 1,0:N − 1]; output: y[0:N − 1]){
    for k = 0 to N − 1 do
        y[k] = 0;                          // initialize y_k
        for l = 0 to N − 1 do
            y[k] = y[k] + A[k, l] * x[l];  // a_kl ≡ A[k, l], and x_l ≡ x[l]
}
```

17

THE COMPUTATIONAL END-USER PERSPECTIVE
-- EXAMPLE --

- Take input of length N=4, and take the matrix A and input x

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \qquad x = \begin{bmatrix} 3 \\ 2 \\ 4 \\ -2 \end{bmatrix}$$

- Then $y = Ax = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 4 \\ -7 \end{bmatrix} = \begin{bmatrix} 2 \\ 12 \\ 8 \\ -10 \end{bmatrix}$

THE COMPUTATIONAL END-USER PERSPECTIVE
-- DATA-INDEPENDENCE --

- The matrix A that specifies the transforms that we will study (like DFT and DCT) is data-independent
  - That means that A does not change when the input x changes
  - But when the size of the input changes (from N to M), then the matrix changes to a matrix of size $M \times M$ so that the multiplication Ax makes sense
  - So the matrix A of a transform depends **on the input size** N but **not on the content** of the input x
- Thus, when we need to be very precise, we adjust the notation slightly:
  - We denote A by A_N when the input size is N, and
  - by A_M when the input size is M (so for input size 8, for example, A is denoted A_8)
- Note: The matrix of the Karhunen-Loeve Transform changes as the input x changes

THE COMPUTATIONAL END-USER PERSPECTIVE
-- TRANSFORM IN ROW FORM--

- In **column form**: $\boldsymbol{y = Ax}$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}, \quad y = Ax = \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix}$$

- In **row form**: $\boldsymbol{y = Ax \Rightarrow y^T = (Ax)^T = x^T A^T \Rightarrow y^T = x^T A^T}$

$$(x_0, x_1, x_2, \dots, x_{N-1}) = (y_0, y_1, y_2, \dots, y_{N-1}) \begin{bmatrix} & \cdots & \\ \vdots & \ddots & \vdots \\ & \cdots & \end{bmatrix}$$

THE COMPUTATIONAL END-USER PERSPECTIVE

-- MATRIX MULTIPLICATION IN BLOCK FORM --

- Let A be an $N \times N$ matrix, and X an $N \times M$ matrix, and $Y = AX$ an $N \times M$ matrix:

$$AX = \begin{bmatrix} a_{11} & \cdots & a_{N1} \\ \vdots & \ddots & \vdots \\ a_{N1} & \cdots & a_{NN} \end{bmatrix} \begin{array}{c} \begin{bmatrix} \boxed{\begin{matrix} X_{11} \\ \vdots \\ X_{N1} \end{matrix}} & \begin{matrix} \cdots \\ \ddots \\ \cdots \end{matrix} & \boxed{\begin{matrix} X_{1M} \\ \vdots \\ X_{NM} \end{matrix}} \end{bmatrix} \\ \begin{matrix} C_1 & \qquad\qquad & C_M \end{matrix} \end{array} = \begin{bmatrix} AC_1 & \cdots & AC_M \end{bmatrix} = \begin{array}{c} \begin{bmatrix} \boxed{\begin{matrix} Y_{11} \\ \vdots \\ Y_{1N} \end{matrix}} & \begin{matrix} \cdots \\ \ddots \\ \cdots \end{matrix} & \boxed{\begin{matrix} Y_{1M} \\ \vdots \\ Y_{NM} \end{matrix}} \end{bmatrix} \\ \begin{matrix} AC_1 & \qquad\qquad & AC_M \end{matrix} \end{array}$$

- If Y is an $N \times M$ matrix and B is an $M \times M$ matrix, and $Z = YB$ is an $N \times M$ matrix, then

$$YB = \begin{array}{c} R_1 \\ \\ R_N \end{array} \begin{bmatrix} \boxed{Y_{11} \quad \cdots \quad Y_{1M}} \\ \vdots \qquad \ddots \qquad \vdots \\ \boxed{Y_{N1} \quad \cdots \quad Y_{NM}} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{M1} \\ \vdots & \ddots & \vdots \\ b_{M1} & \cdots & b_{MM} \end{bmatrix} = \begin{bmatrix} R_1B \\ \vdots \\ R_NB \end{bmatrix} = \begin{array}{c} R_1B \\ \\ R_NB \end{array} \begin{bmatrix} \boxed{Z_{11} \quad \cdots \quad Z_{1M}} \\ \vdots \qquad \ddots \qquad \vdots \\ \boxed{Z_{1N} \quad \cdots \quad Z_{NM}} \end{bmatrix}$$

THE COMPUTATIONAL END-USER PERSPECTIVE
-- 2D TRANSFORMS: TRANSFORM OF AN IMAGE --

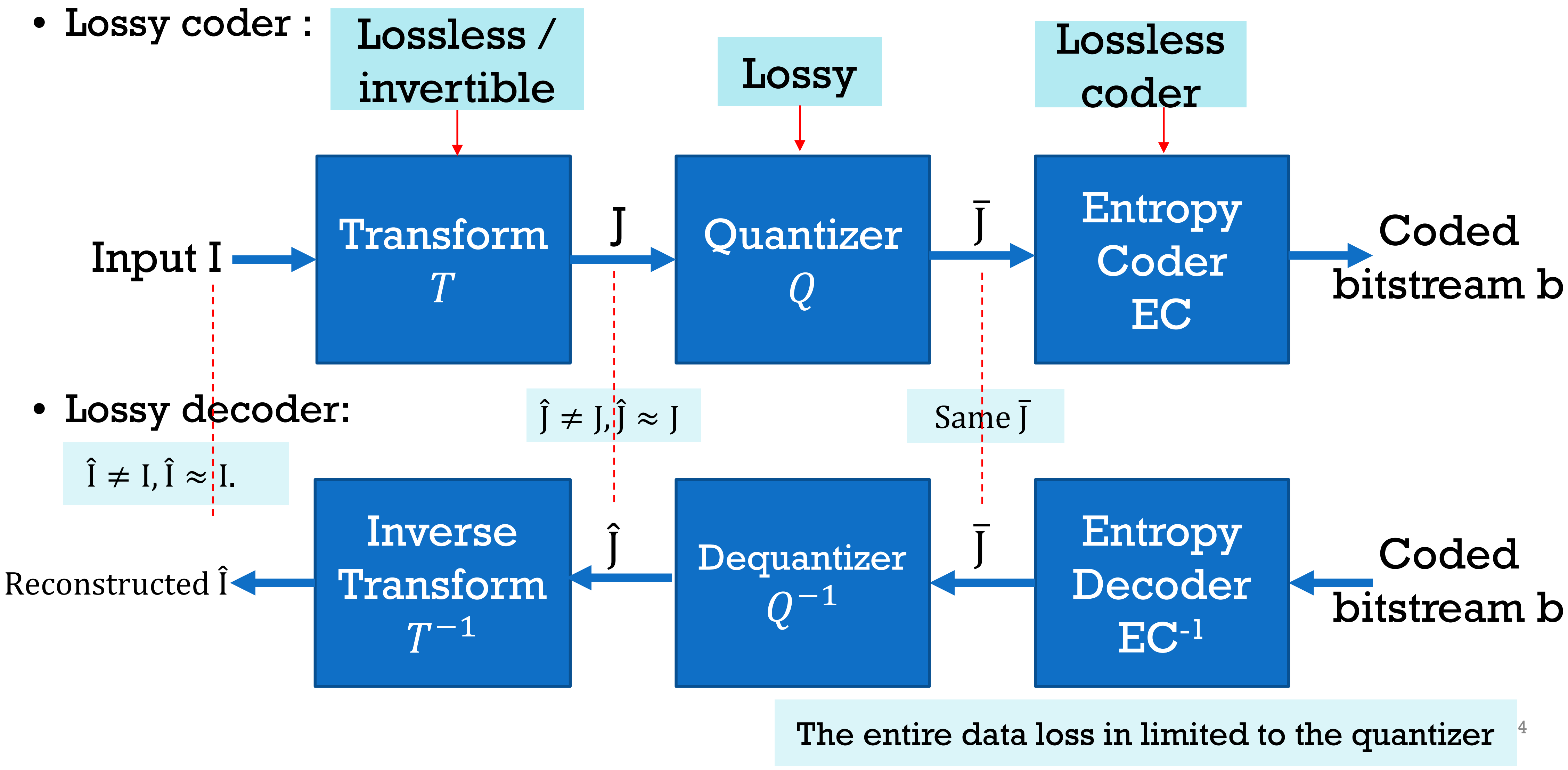
- To transform of an $N \times M$ image X (into another $N \times M$ matrix Z), do :
  1. Transform each column, and then
  2. Transform each row
- In matrix form:
  1. Transform each column of $X = [C_1 \ldots C_M]$, i.e., multiply each column C_i by A on the left:
     a. Compute $AC_1 \quad \cdots \quad AC_M$
     b. Place the resulting columns one after another into a matrix $Y = [AC_1 \quad \cdots \quad AC_M]$
     c. Using last slide, we see that $\boldsymbol{Y = AX = A_N X}$
  2. Transform of each row $Y = \begin{bmatrix} R_1 \\ \vdots \\ R_N \end{bmatrix}$, i.e., multiply each row R_i by A_M^T on the right
     a. Compute $R_1 A_M^T \quad \cdots \quad R_N A_M^T$
     b. Place the resulting rows one above the other into a matrix $\begin{bmatrix} R_1 A_M^T \\ \vdots \\ R_N A_M^T \end{bmatrix}$
     c. Using last slide, we see that $\boldsymbol{Z = Y A_M^T}$

- Combining 1c and 2c, we get: $\boldsymbol{Z = A_N X A_M^T}$

LOSSY COMPRESSION USING TRANSFORMS

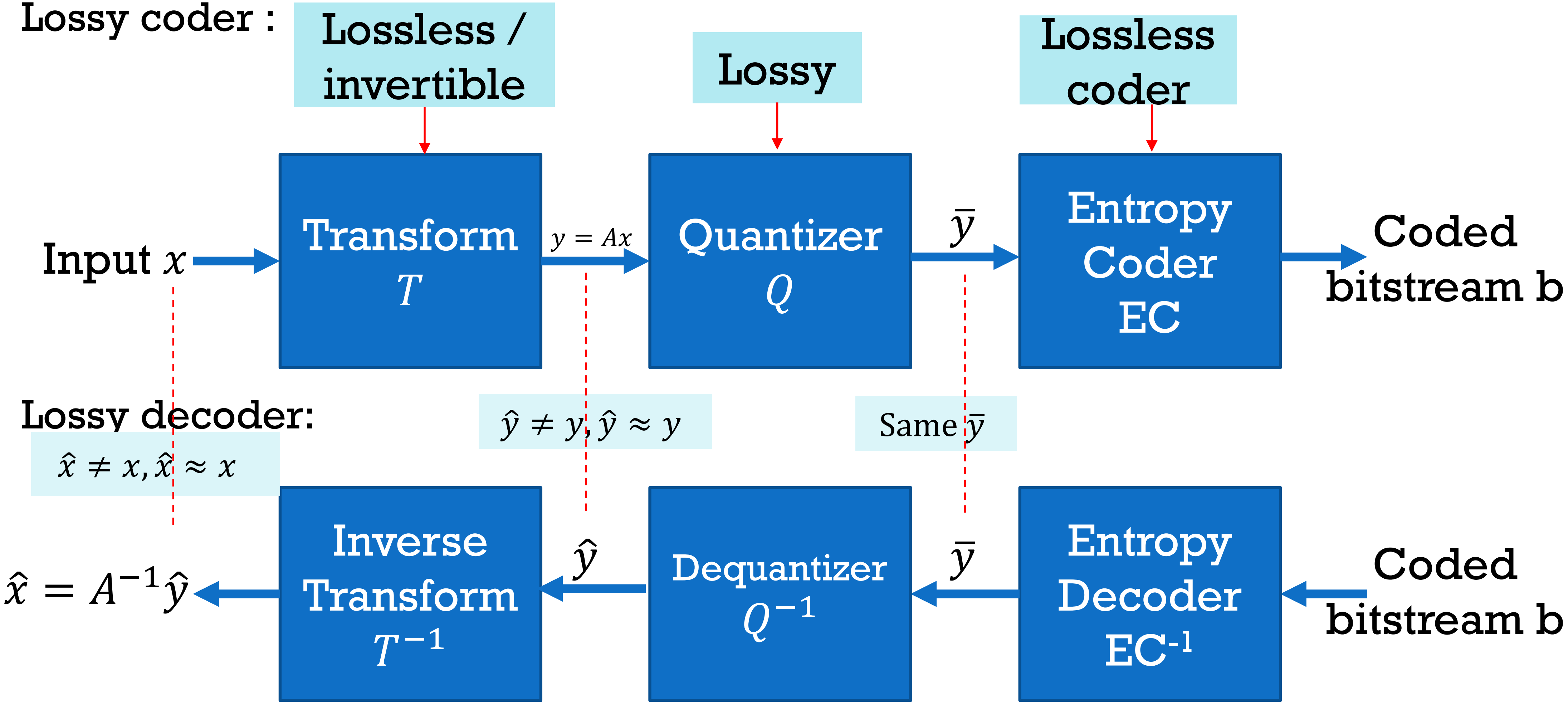
- Let's recall the framework of lossy compression, and link it with transforms

GENERAL SCHEME OF LOSSY COMPRESSION



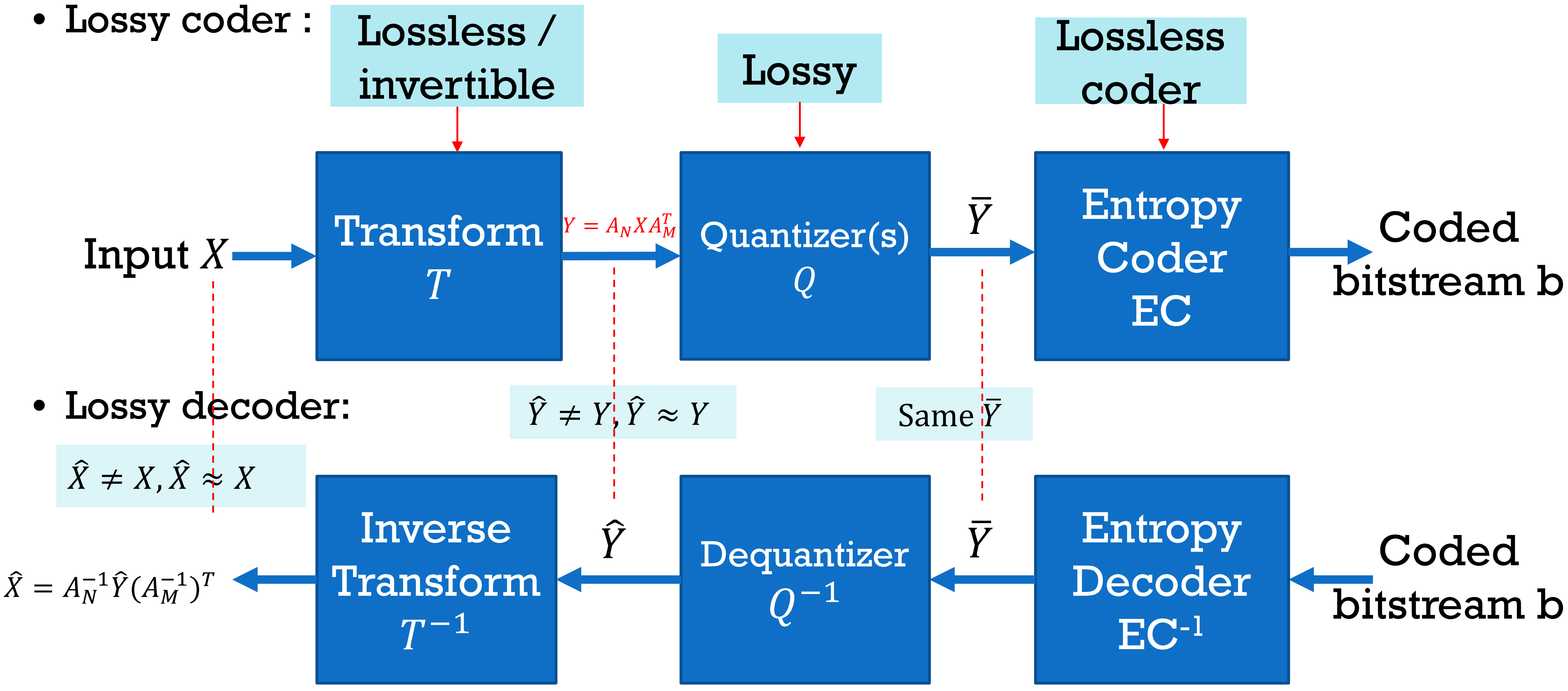
LOSSY COMPRESSION OF 1D SIGNAL USING TRANFORMS

- Lossy coder :
- Lossy decoder:



The entire data loss in limited to the quantizer

LOSSY COMPRESSION OF 2D SIGNAL USING TRANFORMS



The entire data loss in limited to the quantizer

SPEED OF THE TRANSFORM
-- 1D SIGNALS --

- For 1D input signals $x^T = (x_0, x_1, x_2, \dots, x_{N-1})$, and taking the notation $A_N = (a_{kl}), 0 \leq k \leq N-1, 0 \leq l \leq N-1$, and $y = A_N x$ where $y^T = (y_0, y_1, y_2, \dots, y_{N-1})$, we have for each k:
  - $y_k = a_{k0}x_0 + a_{k1}x_1 + a_{k2}x_2 + \cdots a_{k,N-1}x_{N-1}$
  - Thus, y_k takes N multiplications and $N-1$ additions, i.e., O(N) time to compute
  - Since y has N components, computing all of y takes O(N^2) time
- For N large (like in the millions in case of sound signals), O(N^2) time is very slow, especially on small devices
  - For example, DFT needs to run on small devices such as radios and digital phones
  - On a device that can do 1MFLOPS, and N=1 million, O(N^2) =11.5 days!!!! (too long)
  - On a device that can do 1GFLOPS, and N=1 million, O(N^2) =17 minutes (still slow)

SPEED OF THE TRANSFORM
-- 2D SIGNALS --

- For 2D input signals, i.e., an $N \times M$ image X
- Recall that $\boldsymbol{Y = A_N X A_M^T}$ (i.e., we transform every column then every row)
- Every column of A takes O(N^2) time to compute, and so the M columns take O(MN^2) time
- Every row takes O(M^2) time to compute, and so the N columns take O(NM^2) time
- Thus, the 2D transform takes O($MN^2 + \mathrm{NM}^2$)= O(N^3) for $N = M$
- For N=1000, O(N^3)=1 second on a 1GFLOPS device (or 17 mins on a 1MFLOPS)
- But even on a 1GFLOPS device, applying the 2D transform on all the frames of a 2-hour video (with a 30fps rate) takes 2.5 days!

SPEED OF THE TRANSFORM
-- HOW TO HANDLE THE SLOW SPEED --

- From the analysis just done, the transforms are quite slow when applied on real-word data
- Can we do something to speed them up?
- Yes:
  - We will see later that in the case of images, the transforms are applied block-wise, i.e, on individual blocks of size 8×8, which makes the transforms quite fast
  - Also, in the case of DFT and DCT, there is a divide-and-conquer algorithm which does the 1D transform in O$(N \log N)$ time instead of O(N^2)
- **Exercise**: work out the same examples (for N=1 million and N=1,000), to translate O$(N \log N)$ time per column into real time, for 1D and 2D transforms.

MATRICES OF THE TRANSFORMS OF INTEREST

- We will next present the matrices of the transforms of interest
- That is, we will give the matrices A_N of the following transforms
    - The Discrete Fourier Transform (DFT)
    - The Discrete Cosine Transform (DCT)
    - The Hadamard Transform
    - The Walsh Transform
    - The Haar Transform
- The definitions will be given in the form of a_{kl}, where
    - a_{kl} is the generic term in the k^{th} row and l^{th} column position of A_N
    - for all $k = 0, 1, \dots, N-1$ and all $l = 0, 1, \dots, N-1$
    - a_{kl} is a function of k, l, and N, but otherwise does not depend on the input data

THE MATRIX OF THE FOURIER TRANSFORM

- $a_{kl} = \sqrt{\frac{1}{N}} e^{-\frac{2\pi i}{N} kl}$, where i is the complex (imaginary) number $\sqrt{-1}$
- To understand this better, we'll need to review complex numbers
- We'll do that later, after we present the matrices of the other transforms

THE MATRIX OF THE DISCRETE COSINE TRANSFORM

- $a_{0l} = \sqrt{\frac{1}{N}}$ for all $l = 0, 1, \ldots, N-1$;
- $a_{kl} = \sqrt{\frac{2}{N}} \cos \frac{\left(l+\frac{1}{2}\right)k\pi}{N} \ \forall\, k = 1,2,\ldots,N-1, \text{and } l = 0,1,\ldots,N-1$
- $A_N^{-1} = A_N^T$
- Illustrations of the DCT matrix :

$$A_2 = \sqrt{\frac{1}{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, A_4 = \frac{1}{2} \begin{bmatrix} \frac{1}{\sqrt{1+\frac{\sqrt{2}}{2}}} & \frac{1}{\sqrt{1-\frac{\sqrt{2}}{2}}} & \frac{1}{-\sqrt{1-\frac{\sqrt{2}}{2}}} & \frac{1}{-\sqrt{1+\frac{\sqrt{2}}{2}}} \\ \frac{1}{\sqrt{1-\frac{\sqrt{2}}{2}}} & \frac{-1}{-\sqrt{1+\frac{\sqrt{2}}{2}}} & \frac{-1}{\sqrt{1+\frac{\sqrt{2}}{2}}} & \frac{1}{-\sqrt{1-\frac{\sqrt{2}}{2}}} \end{bmatrix}$$

THE MATRIX OF THE HADAMARD TRANSFORM

- To express a_{kl}, we need to express k and l in binary (using $n = \log N$ bits)
  - $k = k_{n-1}k_{n-2} \ldots k_1 k_0, \quad l = l_{n-1}l_{n-2} \ldots l_1 l_0$
  - $a_{kl} = \sqrt{\frac{1}{N}}(-1)^{k_{n-1}l_{n-1}+k_{n-2}l_{n-2}+\cdots+k_1 l_1+ k_0 l_0}$
- $A_N^{-1} = A_N^T = A_N$
- Illustrations:

$$A_2 = \sqrt{\tfrac{1}{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad A_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

$$A_8 = \sqrt{\tfrac{1}{8}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

AN ALTERNATIVE DEFINITION OF THE MATRIX OF THE HADAMARD TRANSFORM

- The Hadamard matrix can be defined recursively (where N is a power of 2, i.e., $N = 2^n$ for some positive integer n)
- $A_1 = (1)$ and $\forall N > 1, A_N = \sqrt{\frac{1}{2}} \begin{pmatrix} A_{\frac{N}{2}} & A_{\frac{N}{2}} \\ A_{\frac{N}{2}} & -A_{\frac{N}{2}} \end{pmatrix}$
- **Exercise**: Derive A_2, A_4 and A_8 using this recursive definition
- **Exercise**: Compare the values A_2, A_4 and A_8 with their values on the previous slide to verify that the two definitions are equivalent
- **Exercise**: Using the recursive definition of A_N, prove by induction on n that A_N is a symmetric matrix, and that $A_N A_N = I_N$, i.e., $A_N^{-1} = A_N$.
- **Note**: In Matlab, if you call hadamard(N), it returns to you the Hadamard matrix A_N but without the constant multiplier $\sqrt{\frac{1}{N}}$

THE MATRIX OF THE WALSH TRANSFORM

- Like in Hadamard, we need to express k and l in binary (using $n = \log N$ bits)
  - $k = k_{n-1}k_{n-2} \dots k_1 k_0, \quad l = l_{n-1}l_{n-2} \dots l_1 l_0$
  - $a_{kl} = \sqrt{\frac{1}{N}}(-1)^{k_{n-1}l_0 + k_{n-2}l_1 + \cdots + k_1 l_{n-2} + k_0 l_{n-1}}$
- $A_N^{-1} = A_N^T = A_N$
- Illustrations:

$$A_2 = \sqrt{\frac{1}{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad A_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

$$A_8 = \sqrt{\frac{1}{8}}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{pmatrix}$$

RELATION BETWEEN HADAMARD AND WALSH

- By comparing the Walsh matrix with the Hadamard matrix, we can quickly see that by permuting the rows and columns of the Walsh matrix we get the Hadamard matrix
- Actually, from the mathematical definition of a_{kl} in each case:
  - Hadamard: $a_{kl} = \sqrt{\frac{1}{N}}(-1)^{k_{n-1}l_{n-1}+k_{n-2}l_{n-2}+\cdots+k_1l_1+\ k_0l_0}$
  - Walsh: $a_{kl} = \sqrt{\frac{1}{N}}(-1)^{k_{n-1}l_0+k_{n-2}l_1+\cdots+k_1l_{n-2}+\ k_0l_{n-1}}$
  - We can see that for $\forall l$ and k:
    - (column l of Hadamard)=(column l^R of Walsh), where l^R is the reverse binary string of l
    - (row k of Hadamard)=(row k^R of Walsh)

- In all the transforms, the columns are indexed from 0 to N-1
- Also the rows are indexed from 0 to N-1

- Verify: column 3(=$(011)_2$) of Hadamard = column 6(=$(110)_2$) of Walsh

EFFECT OF HADAMARD ~ WALSH

- Since the Walsh matrix is a "permutation" of the Hadamard matrix
  - The two transforms have equivalent effects on the input data, as far as compression is concerned
  - That is, if you transform the same input x by the two transforms, the outputs are re-arrangements of each other (Why?)
  - For that reason, scientists don't study the two transforms separately
  - Instead, they "squash" the two transforms into one transform, called the Walsh-Hadamard Transform, and take the Hadamard matrix to be its matrix
  - In fact, Matlab implements Hadamard but not Walsh

THE MATRIX OF THE HAAR TRANSFORM

- Like in the case of Hadamard and Walsh, the Haar matrix is defined only for N being a power of 2 ($N = 2^n$)

How to get p and q for k where $1 \leq k \leq N-1$:
- Take the largest 2-power $\leq k$
- Call it 2^p (note that $2^p \leq k < 2^{p+1}$
- Let $q = k - 2^p$,
- $2^p \leq k < 2^{p+1} \Rightarrow 0 \leq k - 2^p < 2^{p+1} - 2^p = 2^p$

- $a_{0l} = \sqrt{\frac{1}{N}}$ for all $l = 0, 1, \dots, N-1$;
- $for\ k \geq 1, k = 2^p + q, 0 \leq q \leq 2^p - 1, 0 \leq p \leq n-1$:

$$a_{kl} = \sqrt{\frac{1}{N}} \begin{cases} 2^{\frac{p}{2}}, & \text{if } q2^{n-p} \leq l < \left(q + \frac{1}{2}\right) 2^{n-p} \\ -2^{\frac{p}{2}}, & \text{if} \left(q + \frac{1}{2}\right) 2^{n-p} \leq l < (q+1)2^{n-p} \\ 0, & \text{otherwise} \end{cases}$$

- Ex:, for $N = 2^3 = 8, n = 3$, and for k=3, $k = 2^1 + 1$, so $p = 1, q = 1$,

THE MATRIX OF THE HAAR TRANSFORM
-- EXAMPLE --

$$a_{kl} = \sqrt{\frac{1}{N}} \begin{cases} 2^{\frac{p}{2}}, & \text{if } q2^{n-p} \le l < \left(q + \frac{1}{2}\right) 2^{n-p} \\ -2^{\frac{p}{2}}, & \text{if } \left(q + \frac{1}{2}\right) 2^{n-p} \le l < (q+1)2^{n-p} \\ 0, & \text{otherwise} \end{cases}$$

How to get p and q for k where $1 \le k \le N-1$:
- Take the largest 2-power $\le k$
- Call it 2^p (note that $2^p \le k < 2^{p+1}$
- Let $q = k - 2^p$,
- $2^p \le k < 2^{p+1} \Rightarrow 0 \le k - 2^p < 2^{p+1} - 2^p = 2^p$

- Ex:, for $N = 2^3 = 8, n = 3,$ and for k=3, $k = 2^1 + 1,$ so $p = 1, q = 1$:
  - $q2^{n-p} \le l < \left(q + \frac{1}{2}\right) 2^{n-p} \Rightarrow 4 \le l < 6 \Rightarrow a_{34} = a_{35} = \sqrt{\frac{1}{8}} 2^{\frac{p}{2}} = \sqrt{\frac{1}{8}} \sqrt{2}$
  - $\left(q + \frac{1}{2}\right) 2^{n-p} \le l < (q+1)2^{n-p} \Rightarrow 6 \le l < 8 \Rightarrow a_{36} = a_{37} = -\sqrt{\frac{1}{8}} 2^{\frac{p}{2}} = -\sqrt{\frac{1}{8}} \sqrt{2}$

THE MATRIX OF THE HAAR TRANSFORM
-- ILLUSTRATIONS: A_2, A_4, A_8 --

$$A_8 = \sqrt{\tfrac{1}{8}} \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{2} & \sqrt{2} & -\sqrt{2} & -\sqrt{2} \\ 2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 \end{pmatrix}$$

$$a_{34} = a_{35} = \sqrt{\frac{1}{8}}\sqrt{2}$$

$$a_{36} = a_{37} = -\sqrt{\frac{1}{8}}\sqrt{2}$$

$$A_2 = \sqrt{\tfrac{1}{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \qquad A_4 = \tfrac{1}{2} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{pmatrix}$$

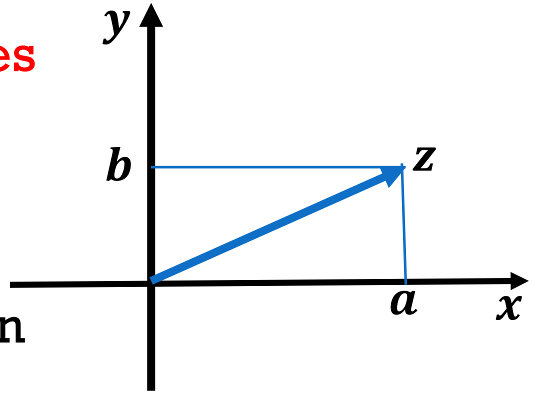
THE MATRIX OF THE FOURIER TRANSFORM

- $a_{kl} = \sqrt{\frac{1}{N}} e^{-\frac{2\pi i}{N} kl}$, where i is the complex (imaginary) number $\sqrt{-1}$
- To understand this better, we'll need to review complex numbers next

COMPLEX NUMBERS
-- DEFINITION AND OPERATIONS --

- Recall that every real number a is a point on the real axis line

- Complex numbers are a generalization of real numbers:
  - Each complex number is a point on the plane (in the x-y coordinate system)
  - Therefore, a complex number is a pair of real number coordinates
  - Notation: a complex number $z = (a, b)$



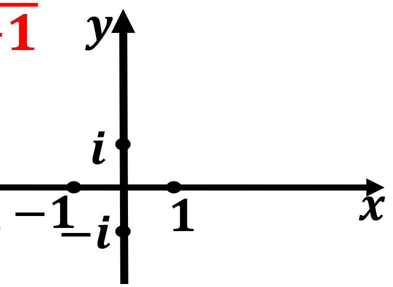
- Operations on complex numbers:
  - Addition (+): $(a, b) + (a', b') \stackrel{\text{def}}{=} (a + a', b + b')$, like vector addition
  - Multiplication (.): $(a, b).(a', b') \stackrel{\text{def}}{=} (aa' - bb', ab' + a'b)$
  - Multiplication by a real number: $c.(a, b) \stackrel{\text{def}}{=} (c.a, c.b)$
- Geometrically, every complex number $(a, 0)$ falls on the x-axis (i.e., the real axis), and therefore coincides with the real number a
- So, we can identify the two: $(a, 0) \equiv a$ (we will use that later)

COMPLEX NUMBERS
-- PROPERTIES OF OPERATIONS --

- $(a,b)+(a',b') \stackrel{\text{def}}{=} (a+a',b+b'); (a,b).(a',b') \stackrel{\text{def}}{=} (aa'-bb',ab'+a'b);\ \text{c}.(\text{a},\text{b})=(\text{ca},\text{cb})$
- It can be shown that the following properties hold:
  1. $(a,b)+(a',b')=(a',b')+(a,b)$ // + is commutative
  2. $((a,b)+(a',b'))+(a'',b'')=(a,b)+((a',b')+(a'',b''))$ // + is associative
  3. $(a,b)+(0{,}0)=(a,b)$ // so (0,0) is like 0
  4. $(a,b).(a',b')$=$(a',b').(a,b)$ // mult is commutative
  5. $((a,b).(a',b')).(a'',b'')=(a,b).((a',b').(a'',b''))$ // mult is is associative
  6. (a,b).(1,0)=(a,b) // so (1,0) is like 1
  7. $(a,b).[(a',b')+(a'',b'')]=(a,b).(a',b')+(a,b).(a'',b'')$ // (.) is distributive over +
  8. If $(a,b)\neq(0{,}0)$, then $(a,b).\left(\frac{a}{a^2+b^2},-\frac{b}{a^2+b^2}\right)=(1{,}0)$ //$\left(\frac{a}{a^2+b^2},-\frac{b}{a^2+b^2}\right)$ is inverse of (a,b)
  9. $c.[(a,b)+(a',b')]=c.(a',b')+c.(a,b), (c+d).(a,b)=c.(a,b)+d.(a,b),$ $c.\big(d.(a,b)\big)=(cd).(a,b)=(cda,cdb)$

COMPLEX NUMBERS
-- ALTERNATIVE NOTATION --

- $(a,b)+(a',b') \stackrel{\text{def}}{=} (a+a',b+b'); (a,b).(a',b') \stackrel{\text{def}}{=} (aa'-bb',ab'+a'b);$ c.(a,b)=(ca,cb)
- (0,1).(0,1)=(0*0-1*1,0*1+1*0)=(-1,0) $\equiv -1$, thus $\mathbf{(0,1)^2=-1}, i.e., \mathbf{(0,1)=\sqrt{-1}}$
- Denote by $\boldsymbol{i=(0,1)}$; // that is a standard notation $\boldsymbol{i^2=-1}$
- Every complex $z=(a,b)=(a,0)+(0,b)=a(1,0)+(0,1)b=a.1+ib=a+ib$

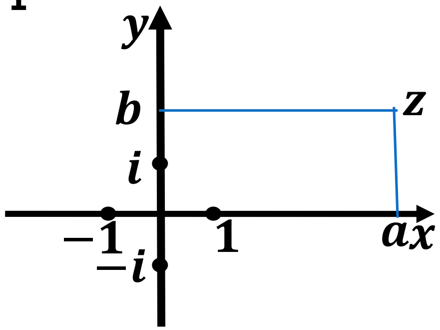


- Therefore, we have an alternative notation that is more commonly used than pairs:

$$\boldsymbol{z=a+ib}$$

a = the ***real part*** of z, $(a=Re(z))$,

b = the ***imaginary part*** of z, $(a=Im(z))$



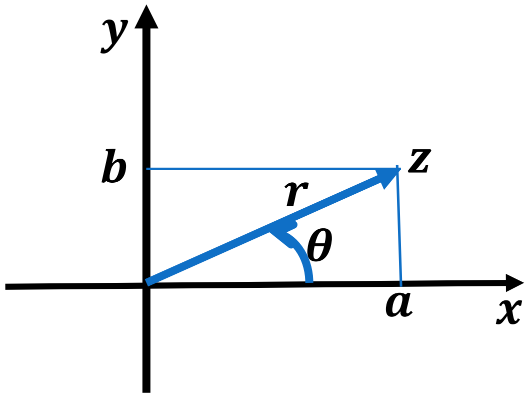
- It is instructive to restate all the 9 properties of the operations using this new notation, especially multiplication:

$$(a+ib).(a'+ib')=aa'-bb'+i(ab'+a'b)$$

- The multiplication rule can be recreated using distributivity and $i^2=-1$

COMPLEX NUMBERS
-- POLAR NOTATION --

- Let's define $re^{i\theta} \stackrel{\text{def}}{=} r\cos\theta + ir\sin\theta$
- What would r and θ be for a given complex number $z = a + ib$?
  - r and θ must satisfy: $\boldsymbol{a = r\cos\theta}$, $\boldsymbol{b = r\sin\theta}$.
  - Therefore, $a^2 + b^2 = r^2(\cos^2\theta + \sin^2\theta) = r^2.1 = r^2$. Thus, $\boldsymbol{r = \sqrt{a^2 + b^2}}$
  - $\frac{b}{a} = \frac{r\sin\theta}{r\cos\theta} = \frac{\sin\theta}{\cos\theta} = \tan\theta$. Thus, $\boldsymbol{\theta = \arctan\frac{b}{a}}$ (for $a \neq 0$)



- Therefore, we call
  - $\boldsymbol{r}$ the ***magnitude*** of z, and denote it $\boldsymbol{r = |z|}$
  - $\boldsymbol{\theta}$ the ***angle*** (or ***phase***) of z, and denote it $\boldsymbol{\theta = Ph(z)}$
- By the way, if $a = 0$, then $\theta = \frac{\pi}{2}$ $if\ b > 0$, and $\theta = -\frac{\pi}{2}$ $if\ b < 0$,

COMPLEX NUMBERS

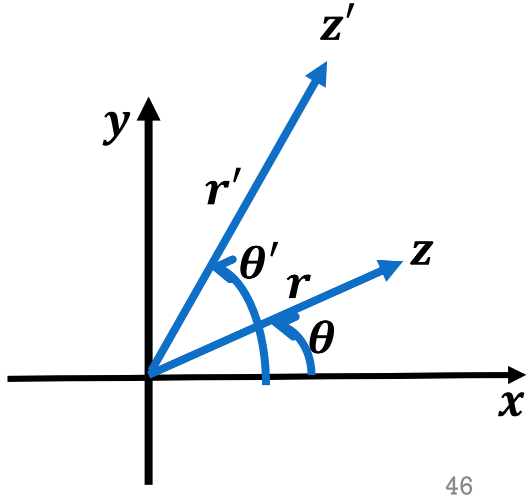
-- POLAR NOTATION: WHY EXPONENTIAL? --

- Let’s define $re^{i\theta} \stackrel{\text{def}}{=} r\cos\theta + ir\sin\theta$
- Let $z = re^{i\theta} \stackrel{\text{def}}{=} r\cos\theta + ir\sin\theta$ and $z' = r'e^{i\theta'} \stackrel{\text{def}}{=} r'\cos\theta' + ir\sin\theta'$
- $z.z' = rr'[\cos\theta\cos\theta' - \sin\theta\sin\theta' + i(\cos\theta\sin\theta' + \sin\theta\cos\theta')]$
- $z.z' = rr'\,[\cos(\theta+\theta') + i\sin(\theta+\theta')]$

From trigonometry:

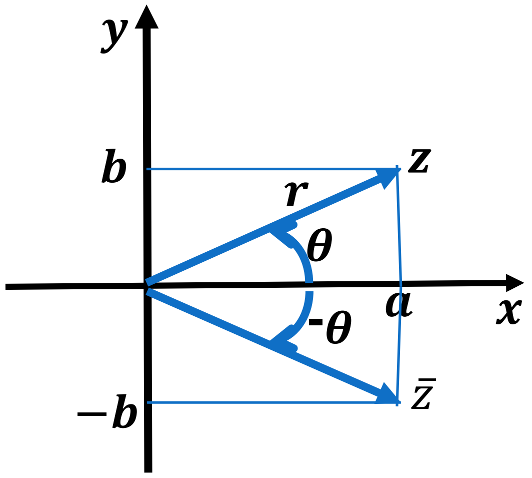
- $\cos(\theta+\theta') = \cos\theta\cos\theta' - \sin\theta\sin\theta'$
- $\sin(\theta+\theta') = \cos\theta\sin\theta' + \sin\theta\cos\theta'$

- Therefore: $\boldsymbol{re^{i\theta}.r'e^{i\theta'} = rr'e^{i(\theta+\theta')}}$
- The above formula justifies the exponential notation
- Geometric explanation of complex multiplication:
  - you add the angles: $\boldsymbol{\theta + \theta'}$
  - you multiply the magnitudes: $\boldsymbol{rr'}$



COMPLEX NUMBERS
-- CONJUGATE AND INVERSE --

- $a + ib + a' + ib' = a + a' + i(b + b'); (a + ib).(a' + ib') = aa' - bb' + i(ab' + a'b);$
- $\mathrm{c.(a + ib)} = ca + icb; re^{i\theta} \stackrel{\text{def}}{=} r\cos\theta + ir\sin\theta; re^{i\theta}.r'e^{i\theta'} = rr'e^{i(\theta+\theta')}$
- Definition: The ***conjugate*** of $z = a + ib$ is $\bar{z} = a - ib$
- $z.\bar{z} = a^2 + b^2 = |z|^2 \Rightarrow \frac{1}{z} = \frac{1}{|z|^2}\bar{z} = \frac{a}{a^2+b^2} - i\frac{b}{a^2+b^2}$
- Also, if $z = re^{i\theta}$, then $\frac{1}{z} = \frac{1}{r}e^{-i\theta} = \frac{1}{r}\cos\theta - \frac{1}{r}\sin\theta$
- So, we can do complex division easily as well



THE MATRIX OF THE FOURIER TRANSFORM

- $a_{kl} = \sqrt{\frac{1}{N}}\, e^{-\frac{2\pi i}{N}kl} = \sqrt{\frac{1}{N}}\left(\cos\frac{2\pi}{N}kl - i\sin\frac{2\pi}{N}kl\right)\ \forall k,l = 0,1,2,\ldots,N-1$
- $A_N^{-1} = \text{conjugate of } A_N$, that is, conjugate of every entry in A_N
- Illustrations of the DFT matrix (where $a = \frac{\sqrt{2}}{2}(1+i)$ and $\bar{a} = \frac{\sqrt{2}}{2}(1-i)$)

$$A_2 = \sqrt{\tfrac{1}{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$A_4 = \frac{1}{2}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

$$A_8 = \sqrt{\frac{1}{8}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \bar{a} & -i & -a & -1 & -\bar{a} & i & a \\ 1 & -i & -1 & i & 1 & -i & -1 & i \\ 1 & -a & i & \bar{a} & -1 & a & -i & -\bar{a} \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -\bar{a} & -i & a & -1 & \bar{a} & i & -a \\ 1 & i & -1 & -i & 1 & i & -1 & -i \\ 1 & a & i & -\bar{a} & -1 & -a & -i & \bar{a} \end{bmatrix}$$

NEXT LECTURE

- Vector spaces
- Vector space perspective of transforms