Fast Fourier Transform

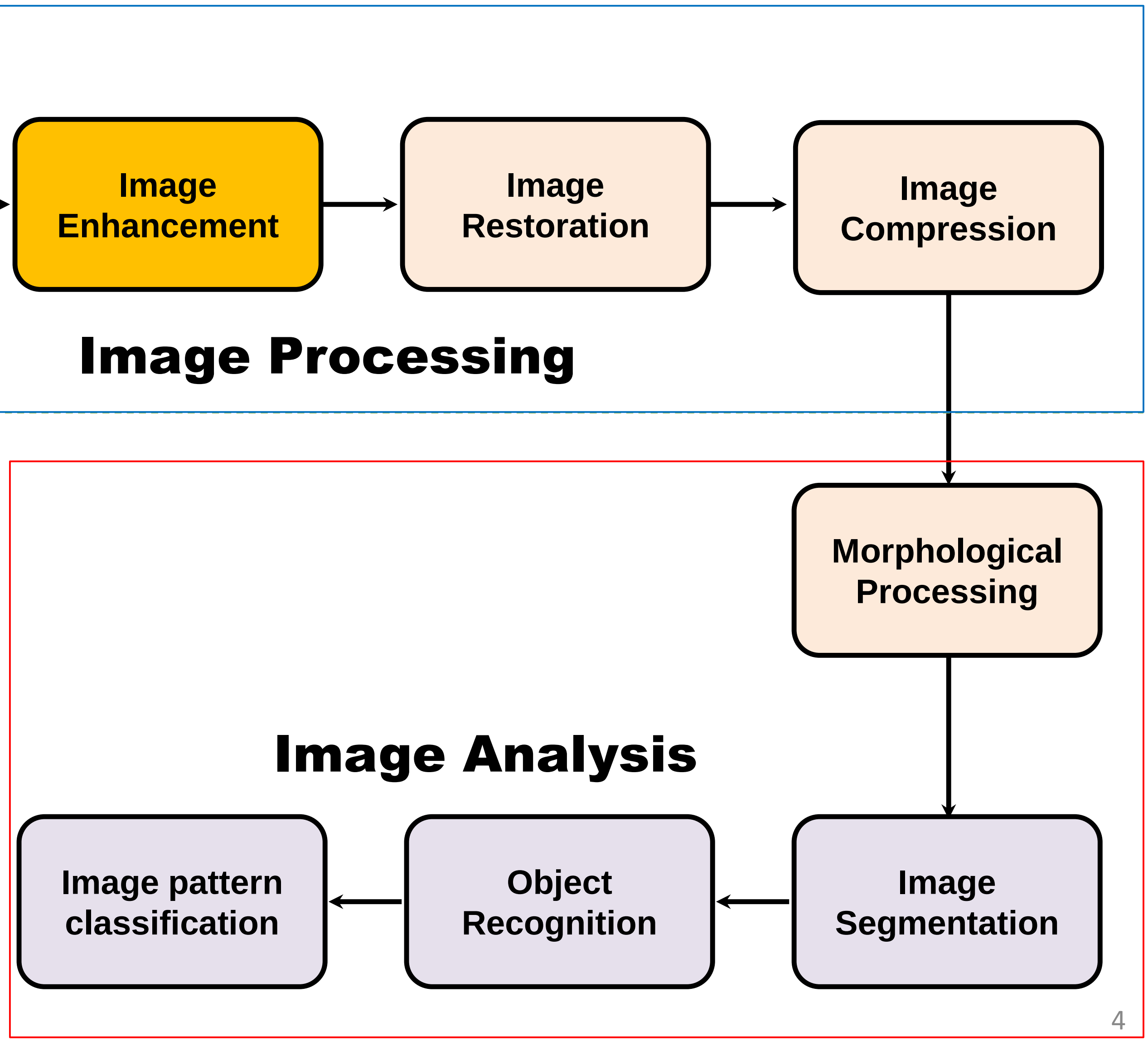
Recap

- Image Smoothing Using Lowpass Frequency Domain Filters
- Image Sharpening Using Highpass Frequency Domain Filters
- Laplacian in the Frequency Domain
- Homomorphic Filtering
- Selective Filtering

Lecture Objectives

- The 2-D DFT - Some Observations
- Separability of Fourier Transform
- IDFT in terms of DFT
- Fast Fourier Transform (FFT)
  - FFT Process in 1-D
  - Special Properties of W_M
  - FFT even-odd approach
  - FFT "Butterfly" Method
  - FFT – time complexity
  - Can we speed it up??
  - FFT Algorithm

Key Stages in DIP



Computer Vision (making sense)

The 2-D DFT - Some Observations

- Let ***f(x,y)*** be a **digital image** of size $M \times N$.

- The two dimensional **DFT pair** is given by:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M + vy/N)}$$

for **u** = 0, 1, 2,…,M–1 and **v** = 0, 1, 2,…, N-1

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux/M + vy/N)}$$

for **x** = 0, 1, 2,…,M–1 and **y** = 0, 1, 2,…, N-1

The 2-D DFT - Some Observations

2-D DFT:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M + vy/N)}$$

for **u** = 0, 1, 2,...,M–1 and **v** = 0, 1, 2,..., N-1

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux/M + vy/N)}$$

for **x** = 0, 1, 2,...,M–1 and **y** = 0, 1, 2,..., N-1

- Computational requirements for implementing 2-D DFT include:
  - **sines and cosine terms**
  - **multiplication**
  - **double summation**

- Brute-force implementation of 2-D DFT and its inverse requires the order of $(MN)^2$ multiplications and additions.

Separability of Fourier Transform

- What properties of ***F(u,v)*** can be useful to **<u>speed up</u>** the calculations?
  - **Separability !!!**

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M + vy/N)}$$

$$= \sum_{x=0}^{M-1} e^{-j2\pi ux/M} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi vy/N}$$

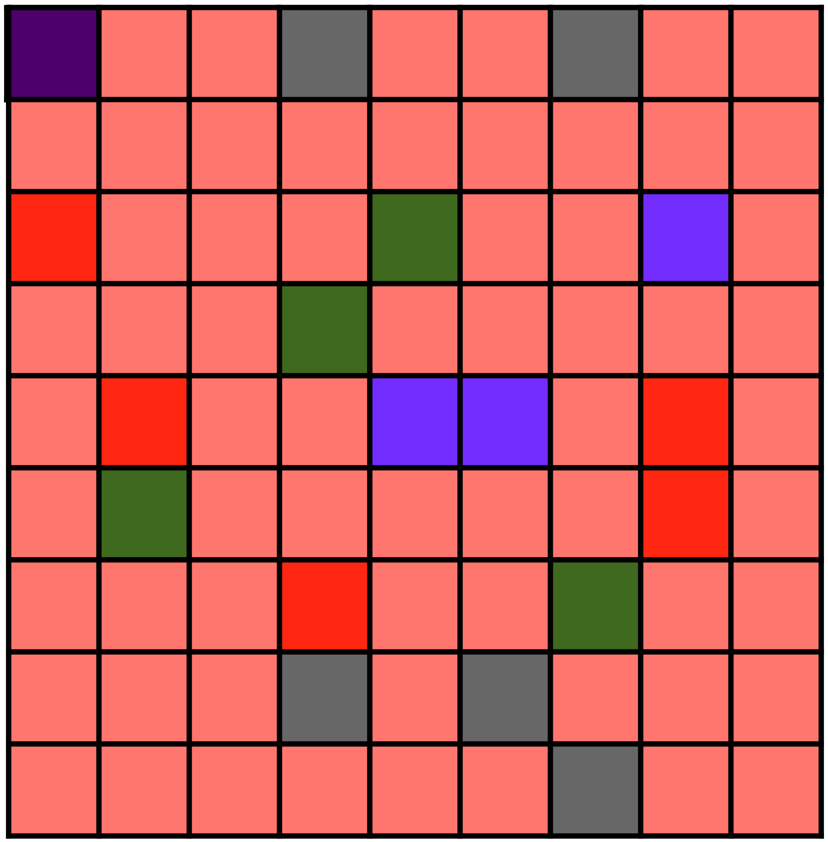
$$= \sum_{x=0}^{M-1} F(x,v) e^{-j2\pi ux/M}$$

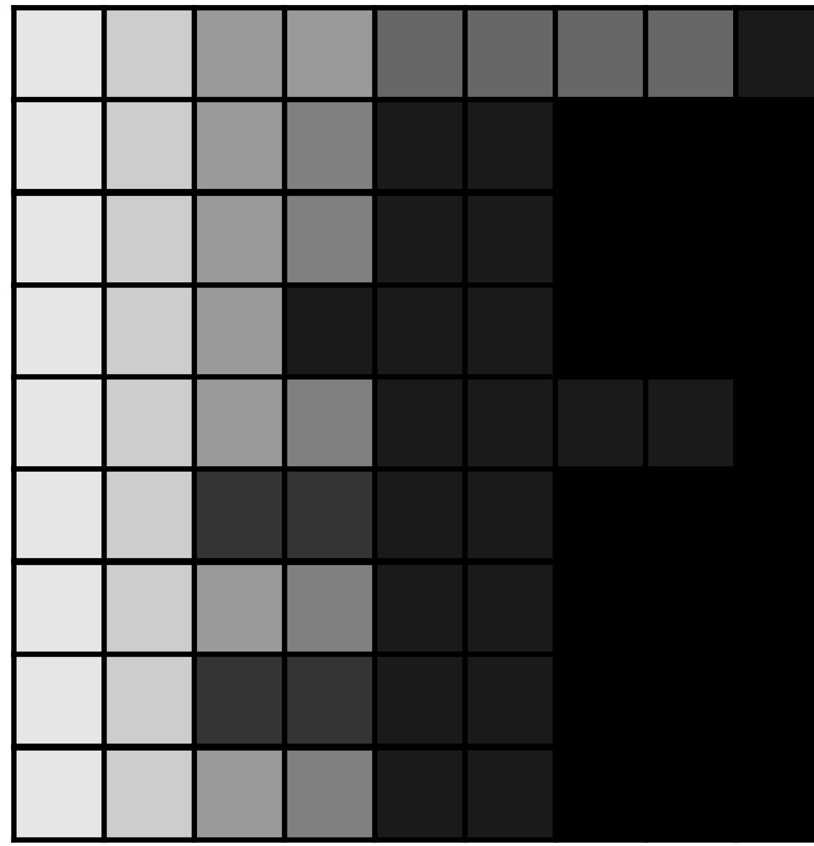
where, $$F(x,v) = \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi vy/N}$$

Separability of Fourier Transform

“Visual Explanation”

First Pass

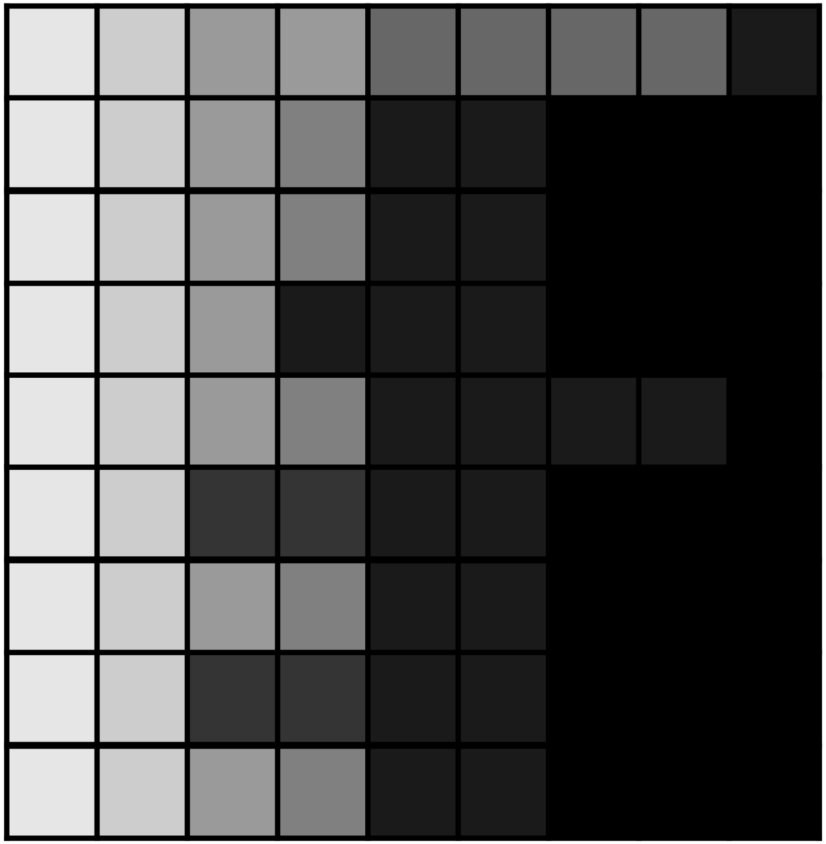




Separability of Fourier Transform

“Visual Explanation”

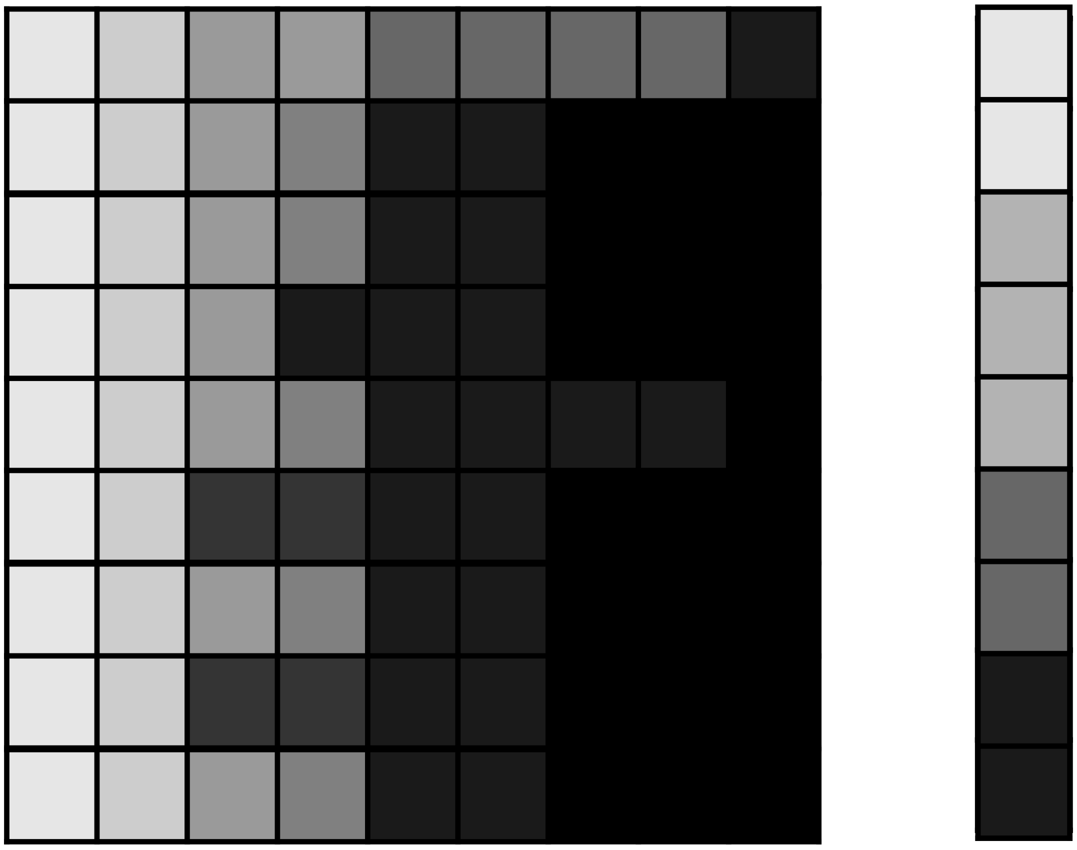
Second Pass



Separability of Fourier Transform

“Visual Explanation”

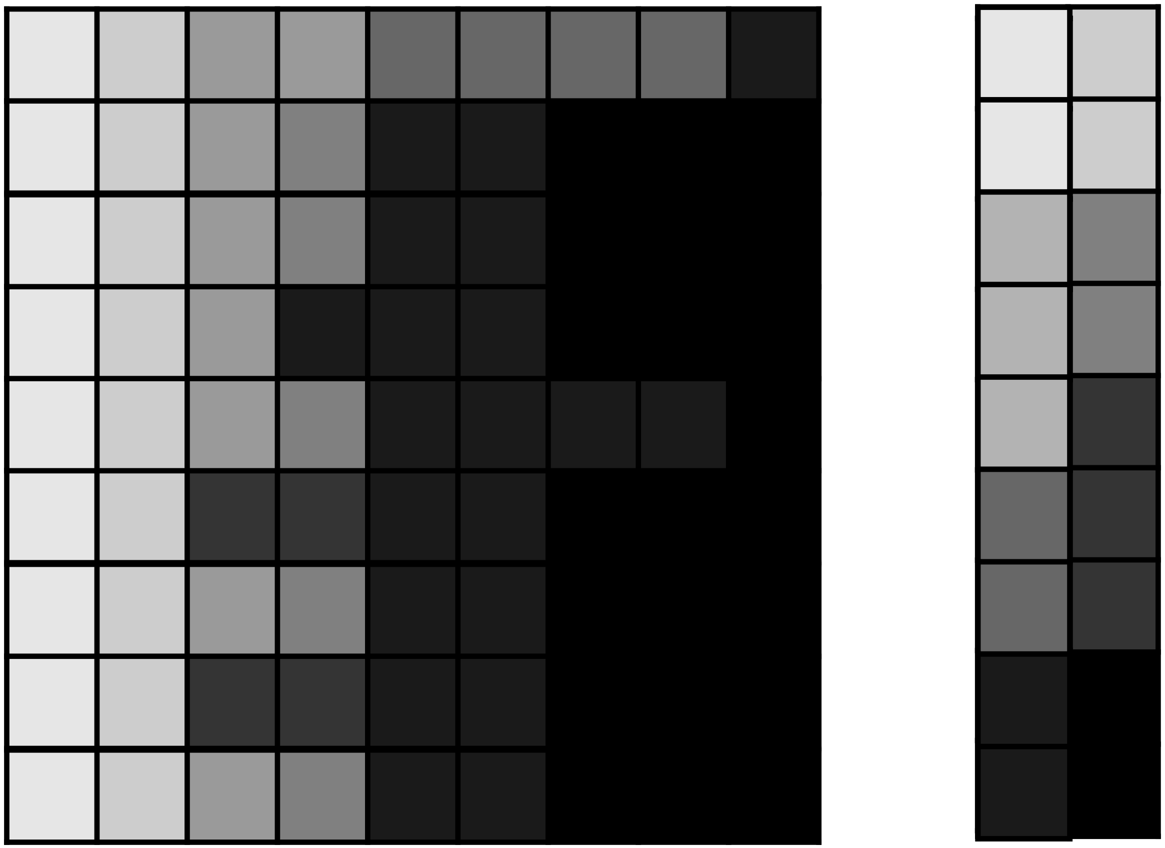
Second Pass



Separability of Fourier Transform

“Visual Explanation”

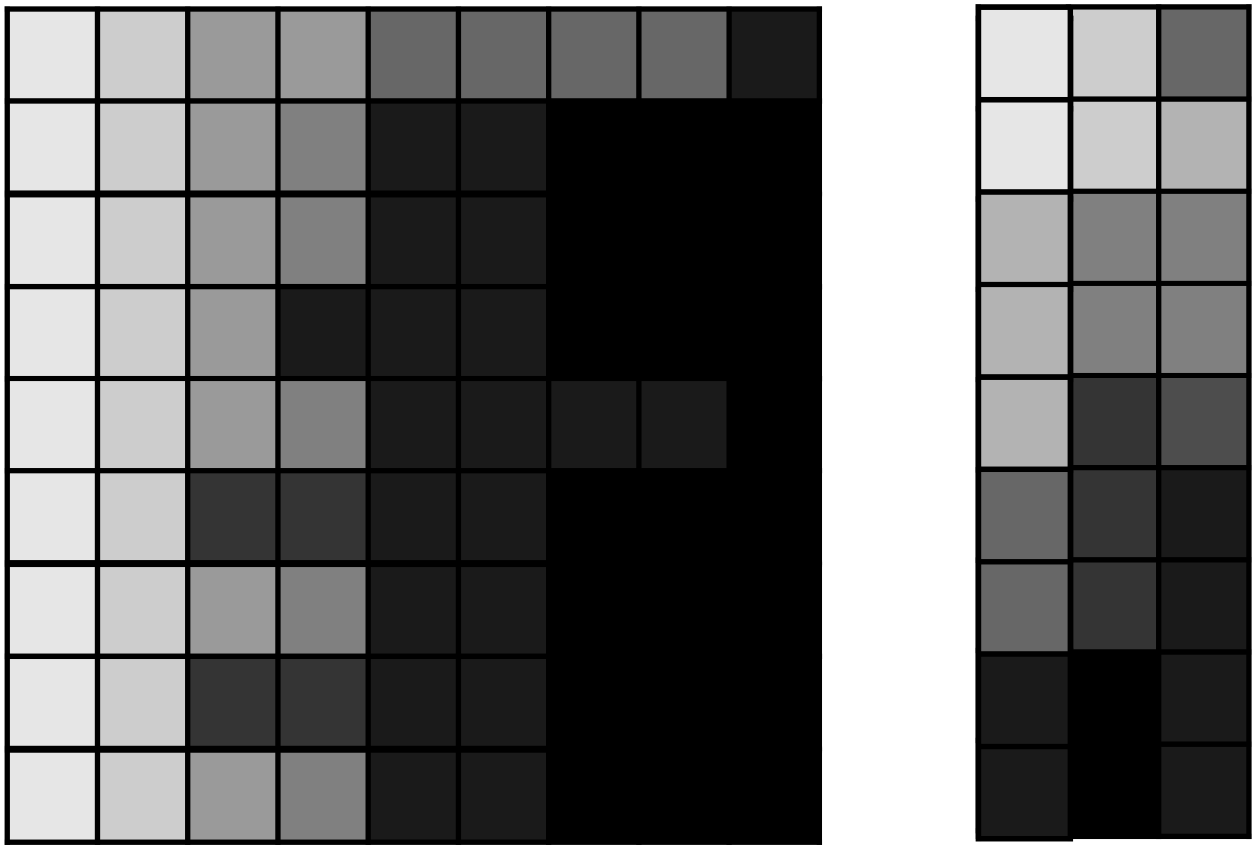
Second Pass



Separability of Fourier Transform

"Visual Explanation"

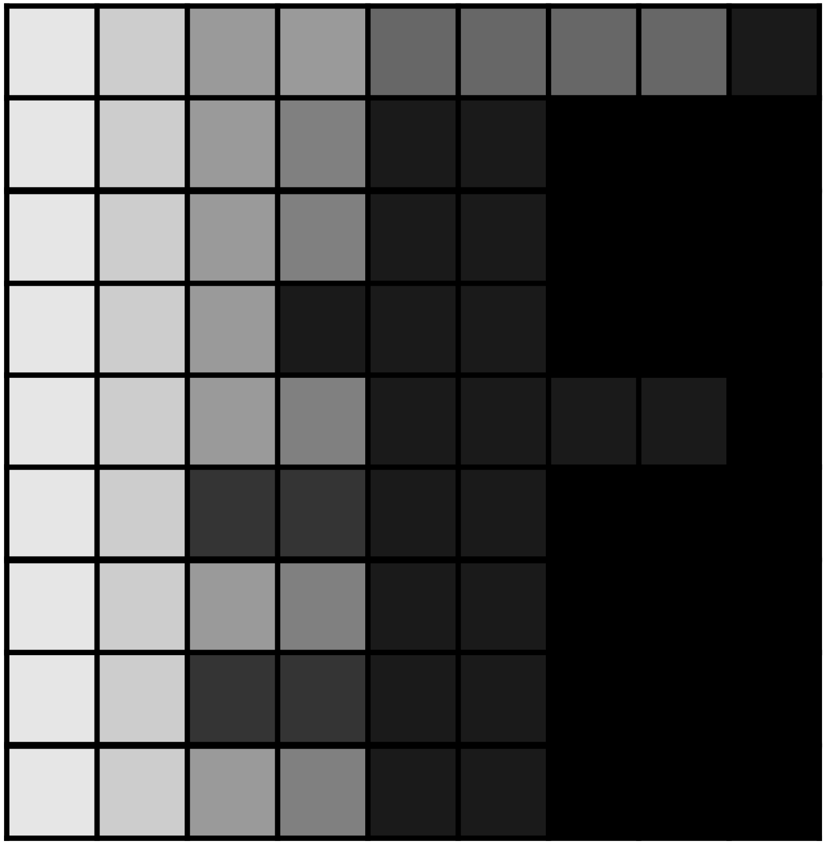
Second Pass

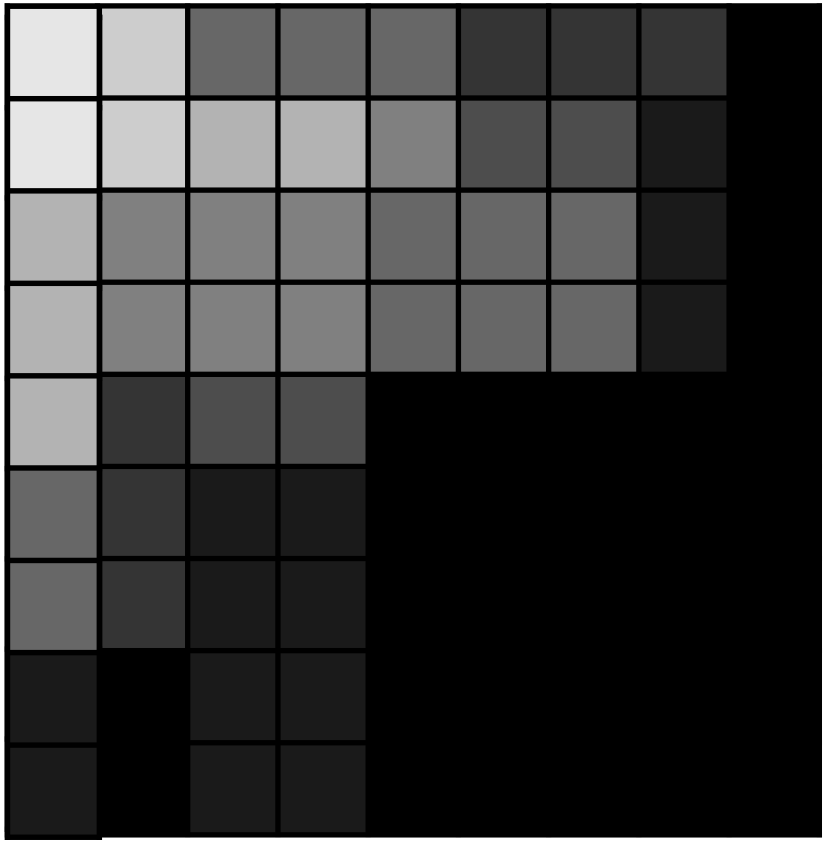


Separability of Fourier Transform

"Visual Explanation"

Second Pass

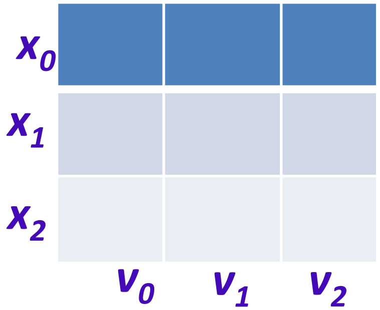




Separability of Fourier Transform

$$F(x,v)=\sum_{y=0}^{N-1} f(x,y)\,e^{-j2\pi vy/N}$$

- For one value of x, and for $v = 0, 1, 2, \dots, N-1$, we see that $F(x,v)$ is the *1-D DFT* of ***one row*** of the image **f(x,y).**



- If we vary x from **0 to M-1**, we have *1D DFT* for ***all rows*** of the image **f(x,y).**

- Similarly, next we compute the *1D DFT* of *these values* for ***all columns*** by varying x from $0, 1, 2, \dots, M-1$ for each value of v from $v = 0, 1, 2, \dots, N-1$

$$= \sum_{x=0}^{M-1} F(x,v)\,e^{-j2\pi ux/M}$$

Thus, we conclude that the 2-D DFT of $f(x,y)$ can be obtained by computing the 1-D transform of **each row of $f(x,y)$** , followed by computing the 1-D transform along **each column of this result**.

Separability of Fourier Transform

"Process So Far"

- We used **two** **1-D** DFT transforms to compute the **2-D** DFT transform of an image.

- What about computing the **IDFT**?
  - We use the property of **complex conjugate** of a Fourier transform to calculate **IDFT** using the standard **DFT equation** !!!

IDFT in terms of DFT

IDFT = $$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)}$$

for **x** = 0, 1, 2,…,M–1 and **y** = 0, 1, 2,…, N-1

Multiply both sides by MN

$$MNf(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M+vy/N)}$$

Take complex conjugate of both sides

$$MNf^*(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F^*(u,v)e^{-j2\pi(ux/M+vy/N)}$$

Given any real number ***x + 0*i**, its **complex conjugate** is ***x - 0*i = *x* itself**.

Given any complex number ***a + bi***, its complex **conjugate** is ***a - bi***.

IDFT in terms of DFT

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M + vy/N)}$$

for **u** = 0, 1, 2,...,M–1 and **v** = 0, 1, 2,..., N-1

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux/M + vy/N)}$$

for **x** = 0, 1, 2,...,M–1 and **y** = 0, 1, 2,..., N-1

$$MNf^*(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F^*(u,v) e^{-j2\pi(ux/M+vy/N)}$$

IDFT in terms of DFT

Steps to obtain $\boldsymbol{f(x,y)}$:

1. Multiply $\mathrm{F(u,v)}$ with MN and take its complex conjugate, $\mathrm{F^*(u,v)}$
2. Perform DFT of $\mathrm{F^*(u,v)}$ for **x** = 0, 1, 2,…,*M*-1 and **y** = 0, 1, 2,…, *N*-1
3. Take the complex conjugate of the result obtained in step-2
4. Divide the result in step-3 by MN --> we have the final result as $\boldsymbol{f(x,y)}$

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(ux/M + vy/N)}$$

$$MNf^*(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F^*(u,v)e^{-j2\pi(ux/M+vy/N)}$$

Separability of Fourier Transform

"Computation cost ??"

- Separable transforms
  - still require operations in the order of **$(MN)^2$**
- Image of size **2048 x 2048**
  - We need order of a **17 trillion** multiplications and additions for **ONE** pass of DFT
    - excluding the exponential terms (sine/cosine) which could be computed once and stored in a look-up table.

Fast Fourier Transform

A Bit of History

- Before FFT was invented, FT was already known for ~150 years and remained as a theoretical analysis tool only.
- In 1965, James Cooley and John Tukey (IBM Watson Research Center) published a paper of FFT.
- Follows Divide-and-conquer strategy.
- It created a boom in DSP and DIP, since FFT can be directly implemented in hardware.

Fast Fourier Transform (FFT)

- Reduces the computational complexity from $\mathbf{(MN)^2}$ to $\mathbf{MN \log_2(MN)}$ operations
  - 2048 x 2048 image
    - Takes order of 92 million operations compared to 17 trillion operations.
    - The difference between $(MN)^2$ to $MN \log_2(MN)$ is immense.
    - With $M = N = 10^6$, for example, it is the difference between, roughly, 30 seconds of CPU time and 2 weeks of CPU time on a microsecond cycle time computer.

- $\mathbf{\log_2}$ should give some idea about the nature of the process
  - Divide-and-Conquer (recursive subdivision into 2 parts)

FFT Process in 1-D

decimation-in-time algorithm

Let $W_M = e^{-j2\pi/M}$

Then, we can express $F(u)$ as :

$$F(u) = \sum_{x=0}^{M-1} f(x)e^{-j2\pi ux/M}$$
$$= \sum_{x=0}^{M-1} f(x)W_M^{ux}$$

We assume $\boxed{M = 2^n}$ for some value of $n \geq 0$

Hence, M can be expressed as $\boxed{M = 2K}$, $K \geq 0$

FFT Process in 1-D

Substituting $M = 2K$ we get

$$F(u) = \sum_{x=0}^{M-1} f(x) W_M^{ux} = \sum_{x=0}^{2K-1} f(x) W_{2K}^{ux}$$

$$= \sum_{x=0}^{K-1} f(2x) W_{2K}^{u(2x)} + \sum_{x=0}^{K-1} f(2x+1) W_{2K}^{u(2x+1)}$$

Split the array of size **2K** into two chunks of size **K** each

Each chunk operates on **ALTERNATE** elements in the original array

FFT Process in 1-D

$$F(u) = \sum_{x=0}^{K-1} f(2x) W_{2K}^{u(2x)} + \sum_{x=0}^{K-1} f(2x+1) W_{2K}^{u(2x+1)}$$

From the definition $W_M = e^{-j2\pi/M}$

$$W_{2K}^{2ux} = e^{-j2\pi(2ux)/(2K)} = e^{-j2\pi ux/K} = W_K^{ux}$$

$$F(u) = \sum_{x=0}^{K-1} f(2x) W_K^{ux} + \sum_{x=0}^{K-1} f(2x+1) W_K^{ux} W_{2K}^{u}$$

Define:

$$F_{even}(u) = \sum_{x=0}^{K-1} f(2x) W_K^{ux}$$

$$F_{odd}(u) = \sum_{x=0}^{K-1} f(2x+1) W_K^{ux}$$

For u = 0, 1, 2, …., k-1

FFT Process in 1-D

$$F(u) = \sum_{x=0}^{K-1} f(2x)W_K^{ux} + \sum_{x=0}^{K-1} f(2x+1)W_K^{ux}W_{2K}^{u}$$

$$F_{even}(u) = \sum_{x=0}^{K-1} f(2x)W_K^{ux}$$

$$F_{odd}(u) = \sum_{x=0}^{K-1} f(2x+1)W_K^{ux}$$

$$F(u) = F_{even}(u) + F_{odd}(u)W_{2K}^{u}$$

$$w_K^{u+K} = w_K^{u}$$

$$w_{2K}^{u+K} = -w_{2K}^{u}$$

Try these equations with **u=2** and **K=2**

$$F(u+K) = F_{even}(u) - F_{odd}(u)W_{2K}^{u}$$

Special Properties of W_M

$$W_M = e^{-j2\pi/M}$$

- The exponential term $\mathbf{W_M}$ has some useful special properties:

1. Symmetric:

$$\boldsymbol{W_M^{k+\frac{M}{2}} = -W_M^k}$$

Example: $W_8^4 = -W_8^0\,, W_8^5 = -W_8^1, W_8^6 = \; -W_8^2, W_8^7 \; -W_8^3$
for **K=0,1,2,3** and **M=8**

2. Periodic:

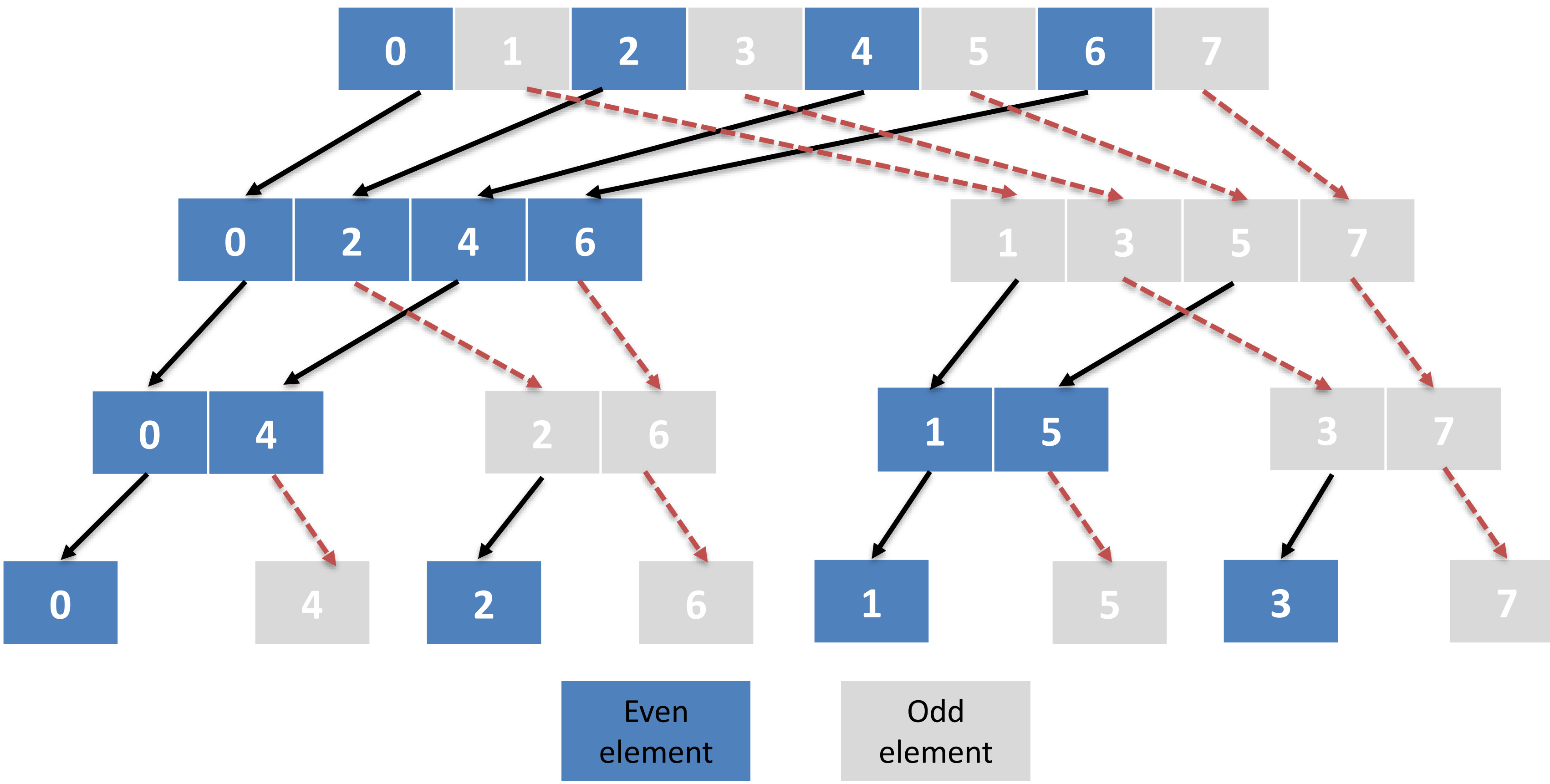
$$\boldsymbol{W_M^{k+M} = W_M^k}$$

Example: $W_8^4 = W_8^{12}$ for **K=4, M=8** , $W_4^2 = W_4^6$ for **K=2, M=4**

FFT even-odd approach

$$F_{even}(u) = \sum_{x=0}^{K-1} f(2x) W_K^{ux}$$

$$F_{odd}(u) = \sum_{x=0}^{K-1} f(2x+1) W_K^{ux}$$



FFT "Butterfly" Method

2-point FFT

- Let us use a simple example with a signal $\boldsymbol{x[n]}$ of length **2**. We have:

$$\boldsymbol{x[n] = x0, x1}$$

where,

- ❑ $\boldsymbol{x0, x1}$ represent the two values of the signal
- ❑ **K=1**
- ❑ **M=2**

$$M = 2K,\ K \geq 0$$

- Use definition of FFT based on the even-odd functions:

$$\boxed{F(u) = F_{even}(u) + F_{odd}(u)W_{2K}^{u}} \quad \boxed{F(u+K) = F_{even}(u) - F_{odd}(u)W_{2K}^{u}}$$

we will have the Fourier Transform of $\boldsymbol{x[n]}$ as:

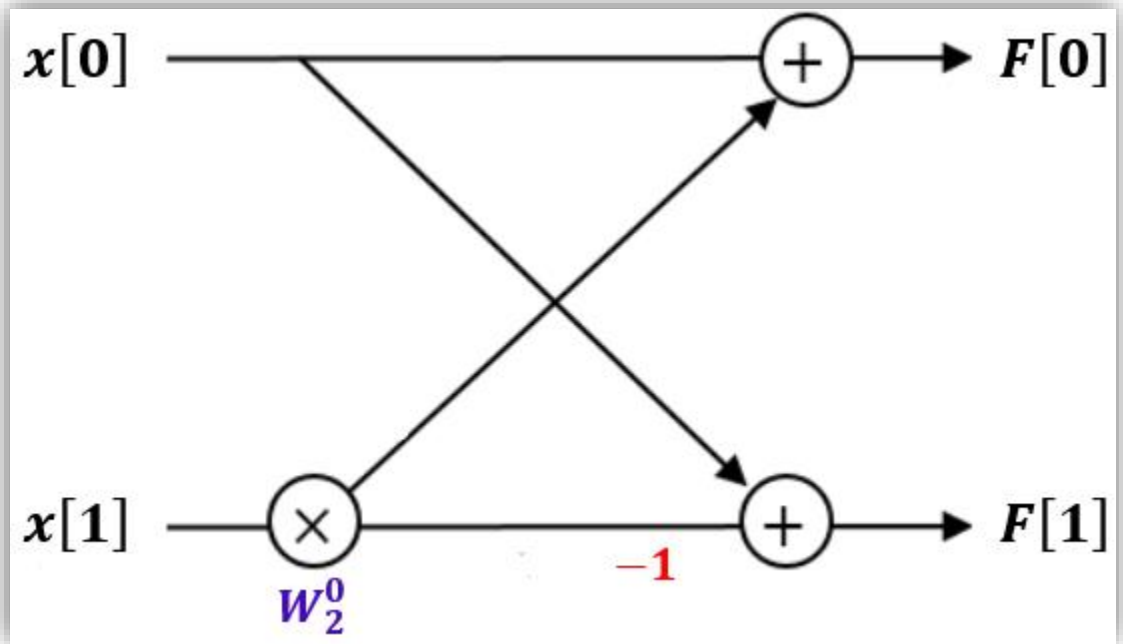
$$\boldsymbol{F(u) = F[0] = x[0] + x[1]\, W_2^0}$$

$$\boldsymbol{F(u+k) = F[1] = x[0] - x[1]\, W_2^0}$$

FFT "Butterfly" Method

2-point FFT

$$F(u) = F[0] = x[0] + x[1]\,W_2^0$$
$$F(u+k) = F[1] = x[0] - x[1]\,W_2^0$$



$\boxed{W_2^0 = 1}$ $\boxed{-W_2^0 = -1}$

FFT "Butterfly" Method

4-point FFT

- Let us use a simple example with a signal $\boldsymbol{x}[\boldsymbol{n}]$ of length **4**. We have:

$$\boldsymbol{x}[\boldsymbol{n}] = \boldsymbol{x0}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}$$

where,

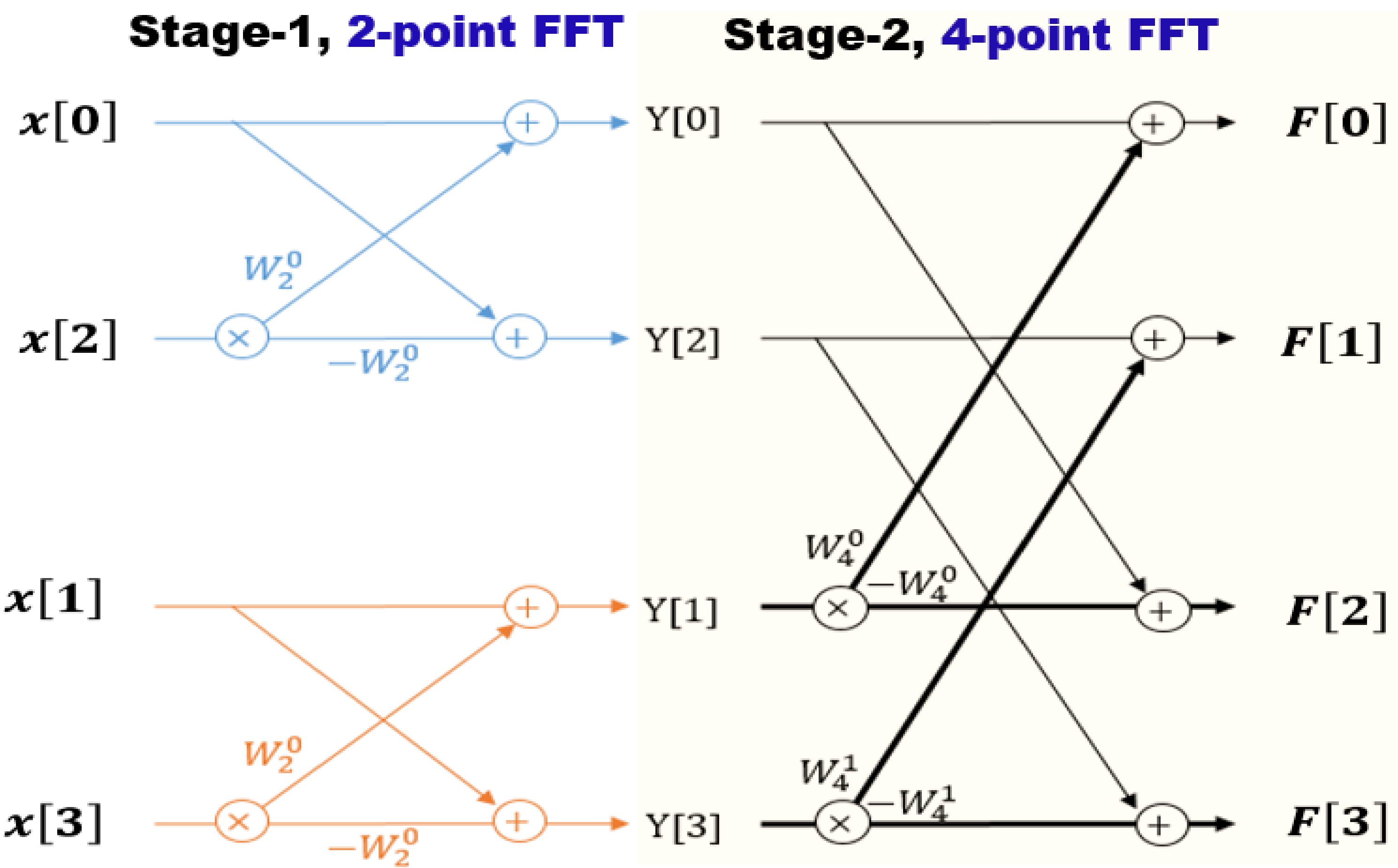
- ❑ $\boldsymbol{x0}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}$ represent the four values of the signal
- ❑ **K=2** $M = 2K, K \geq 0$
- ❑ **M=4**

FFT "Butterfly" Method

4-point FFT

$$F(u) = F_{even}(u) + F_{odd}(u)W_{2K}^{u}$$

$$F(u+K) = F_{even}(u) - F_{odd}(u)W_{2K}^{u}$$



FFT "Butterfly" Method

8-point FFT

- Let us use a simple example with a signal $\boldsymbol{x[n]}$ of length **8**. We have:

$$\boldsymbol{x[n] = x0, x1, x2, x3, x4, x5, x6, x7}$$

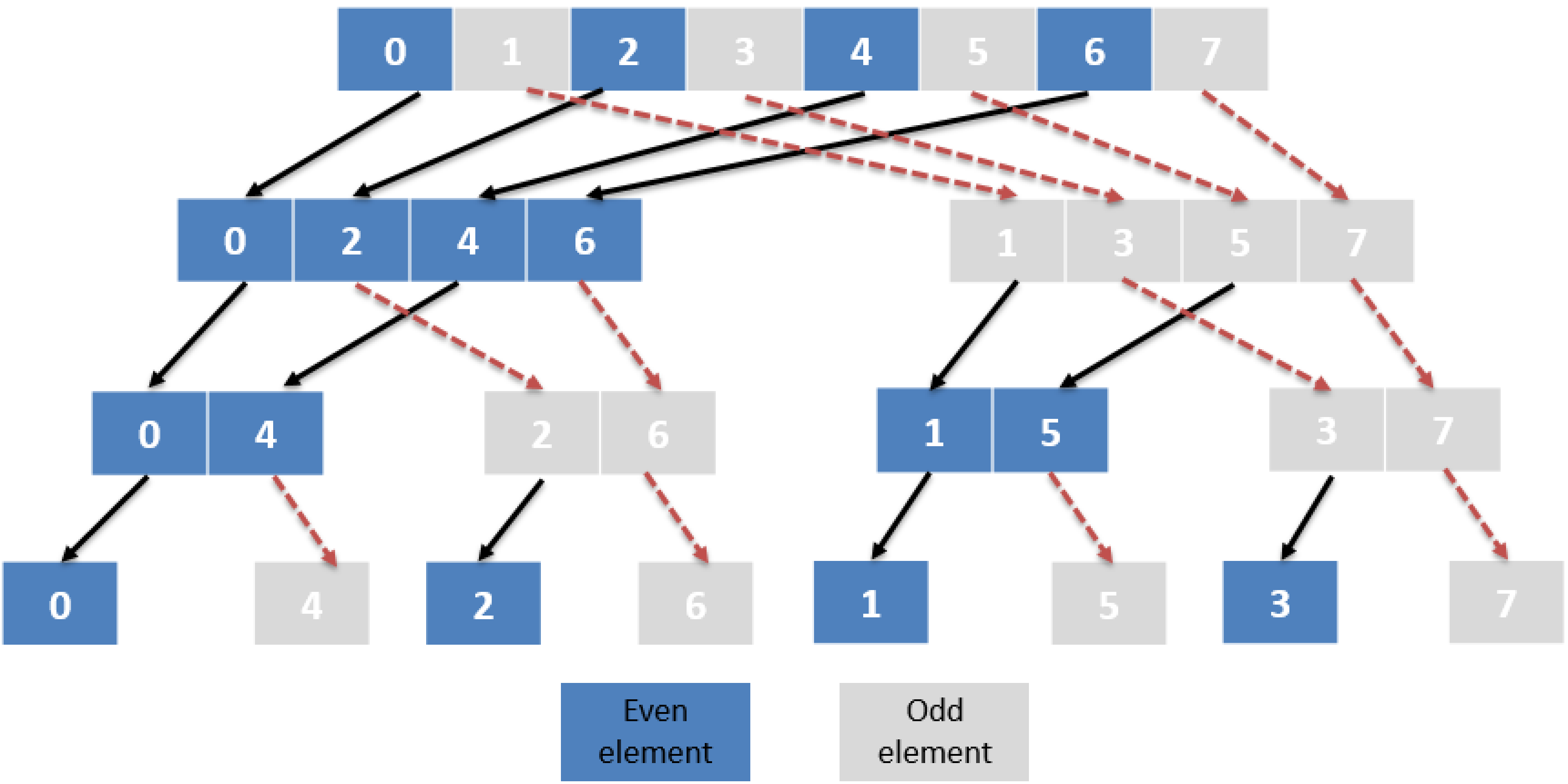
where,

- ❑ $\boldsymbol{x0, x1, x2, x3, x4, x5, x6, x7}$ represent the eight values of the signal
- ❑ **K=4**
- ❑ **M=8**

$$M = 2K,\ K \geq 0$$

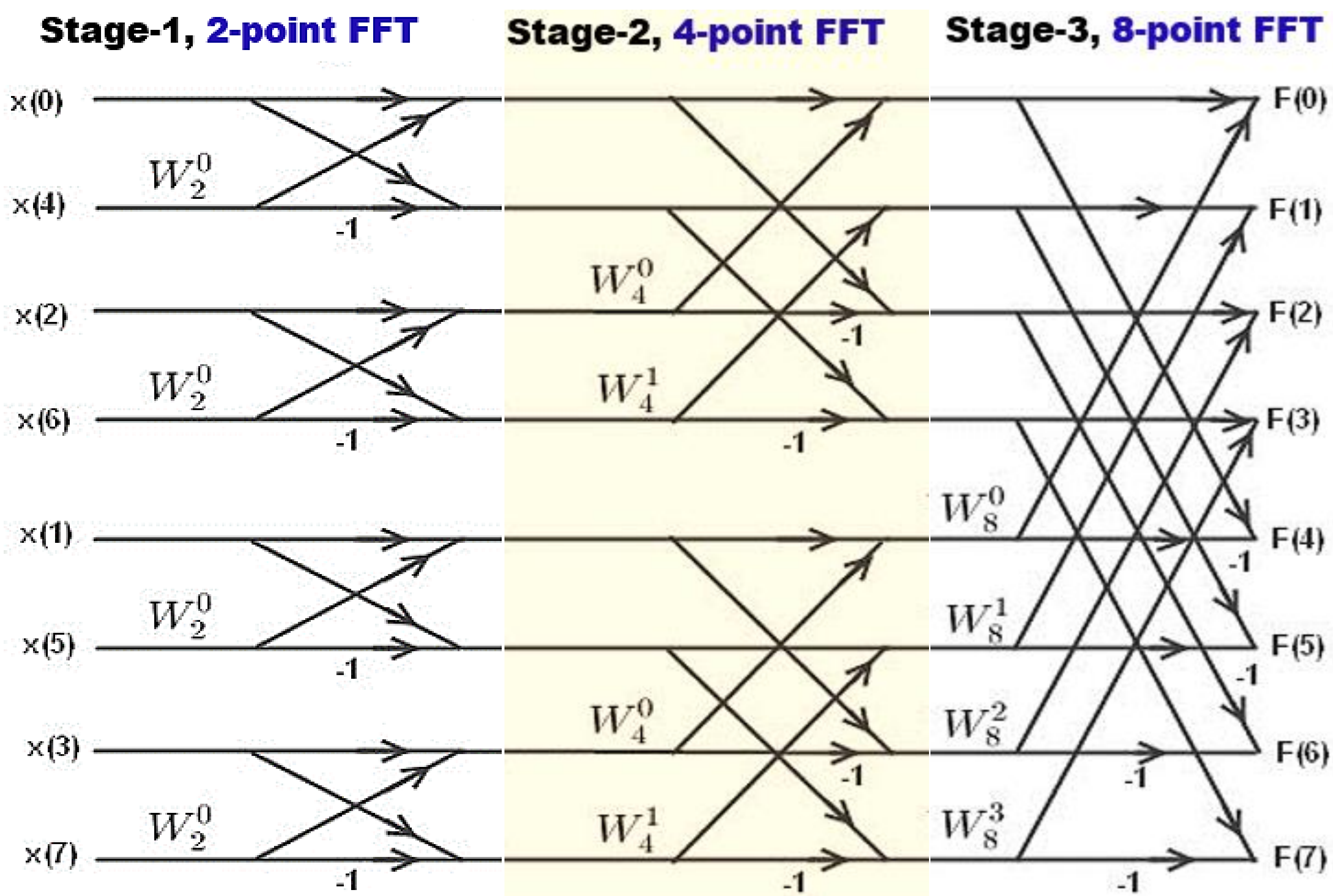
FFT "Butterfly" Method

8-point FFT



FFT "Butterfly" Method

8-point FFT



$W_2^0 = W_4^0 = W_8^0 = 1$ | $W_4^1 = -j$ | $W_8^1 = 0.70 - 0.70j$ | $W_8^2 = -j$ | $W_8^3 = -0.70 - 0.70j$

FFT "Butterfly" Method

16-point FFT

- Let us use a simple example with a signal $\boldsymbol{x}[\boldsymbol{n}]$ of length **16**. We have:

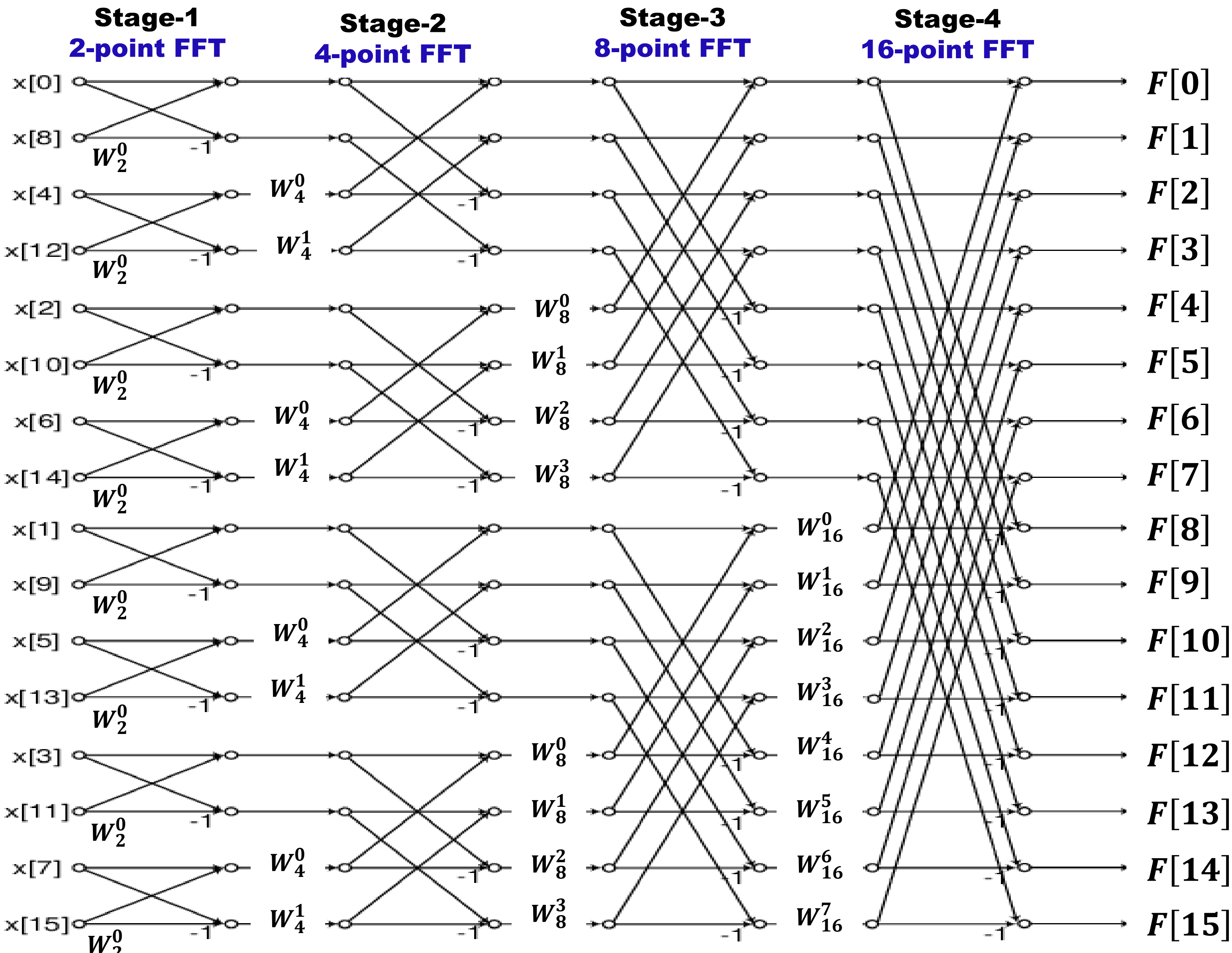
$$\boldsymbol{x}[\boldsymbol{n}] = \boldsymbol{x0}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, \boldsymbol{x4}, \ldots\ldots\ldots, \boldsymbol{x15}$$

where,

- ❑ $\boldsymbol{x0}, \boldsymbol{x1}, \boldsymbol{x2}, \boldsymbol{x3}, \boldsymbol{x4}, \ldots\ldots\ldots, \boldsymbol{x7}$ represent the sixteen values of the signal
- ❑ **K=8** $M = 2K, K \geq 0$
- ❑ **M=16**

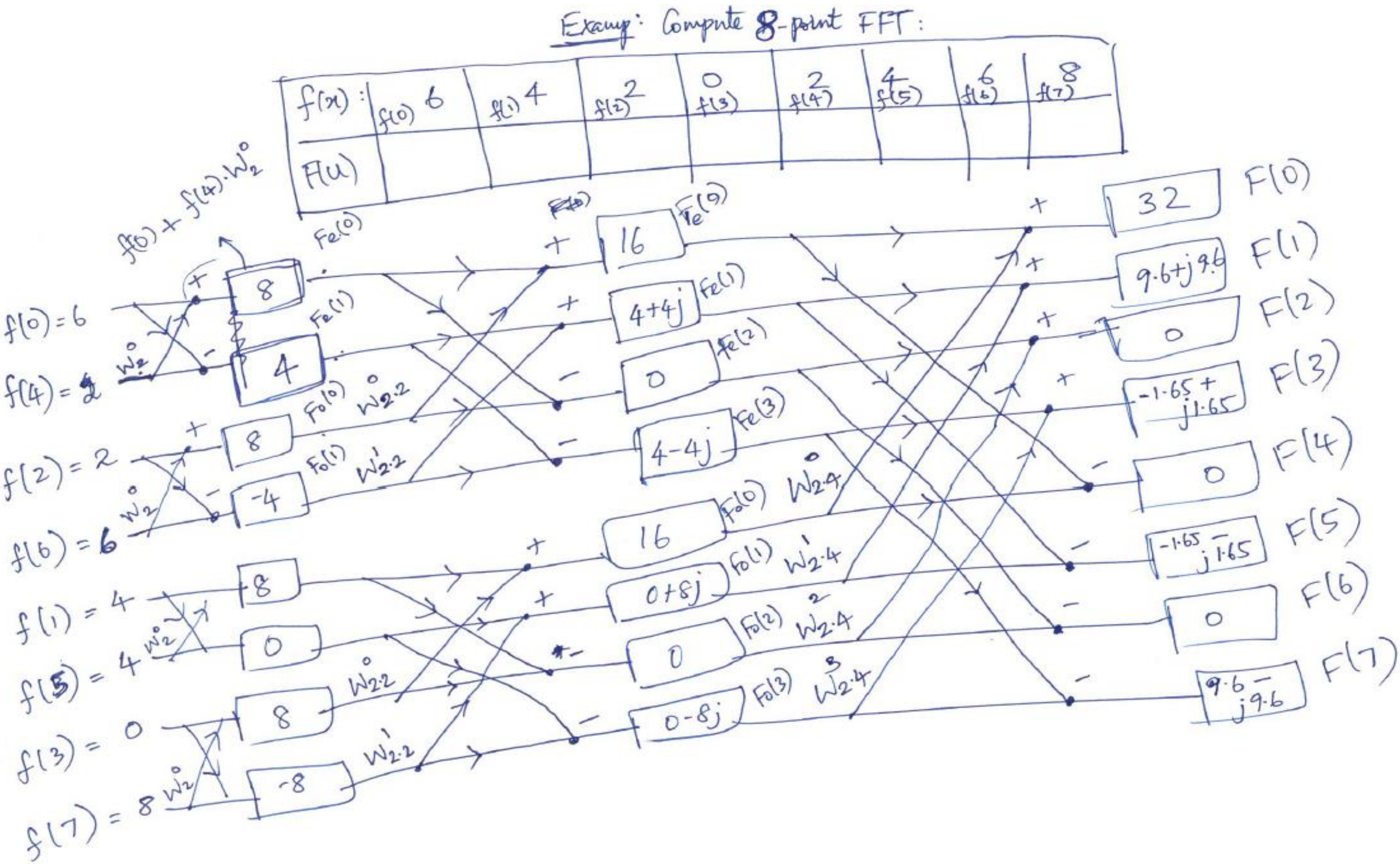
FFT "Butterfly" Method

16-point FFT



FFT "Butterfly" Method

8-point FFT

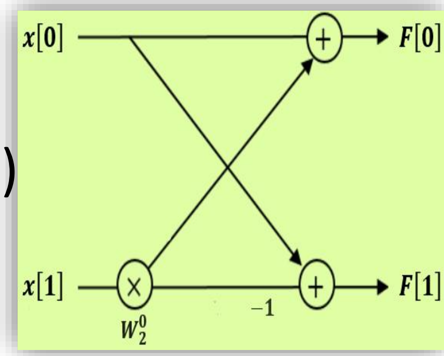


FFT – time complexity

- Every m-point transform can be computed as a sum of **m/2**-point transform.

- Divide the elements into even and odd subsets, and compute the transform for these subsets.

- Can be **2^n** if computed recursively (What is the base case?)



- Total number of operations:
  - Let **m(n)** and **a(n)** represent the number of complex multiplications and complex additions, respectively, where the length of the signal is **2^n**
    - When **K=1**, we need **m(1)=1, a(1)=2** operations:

      $F[0]=x[0] + (x[1] \times W_2^0)$, $F[1]=x[0] - (x[1] \times W_2^0)$
    - When **K=2**, we need **m(2)=2m(1)+2, a(2)=2a(1)+4** operations
    - When **K=3**, we need **m(3)=2m(2)+4, a(3)=2a(2)+8** operations

      …………
    - When **K=n**, we need **$m(n)=2m(n-1)+2^{n-1}$**, **$a(n)=2a(n-1)+2^n$**

$m(n)=2^{(n-1)}$, $a(n)=2^n$

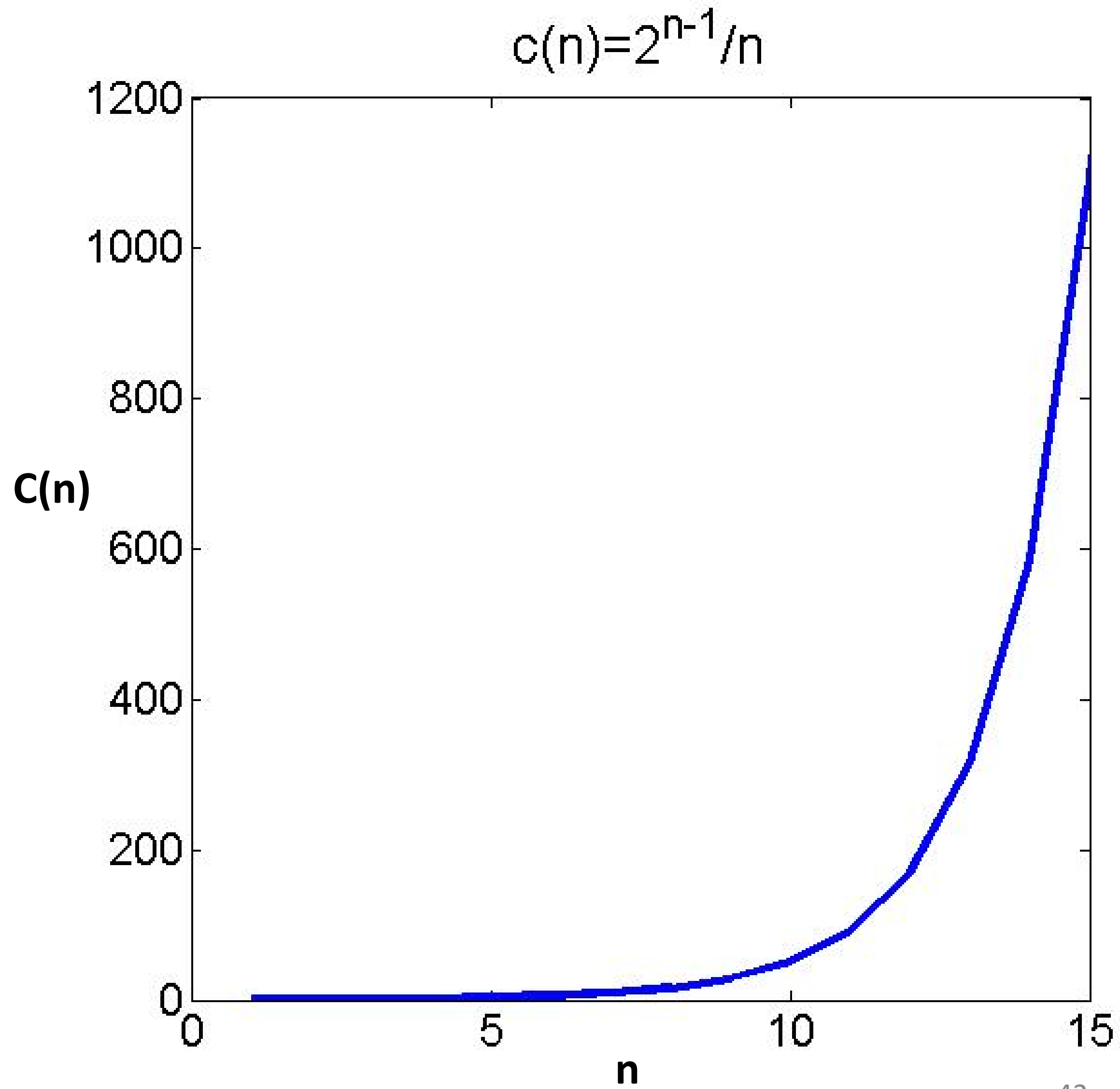
2^n is number of samples in FFT

Computational Advantage of FFT

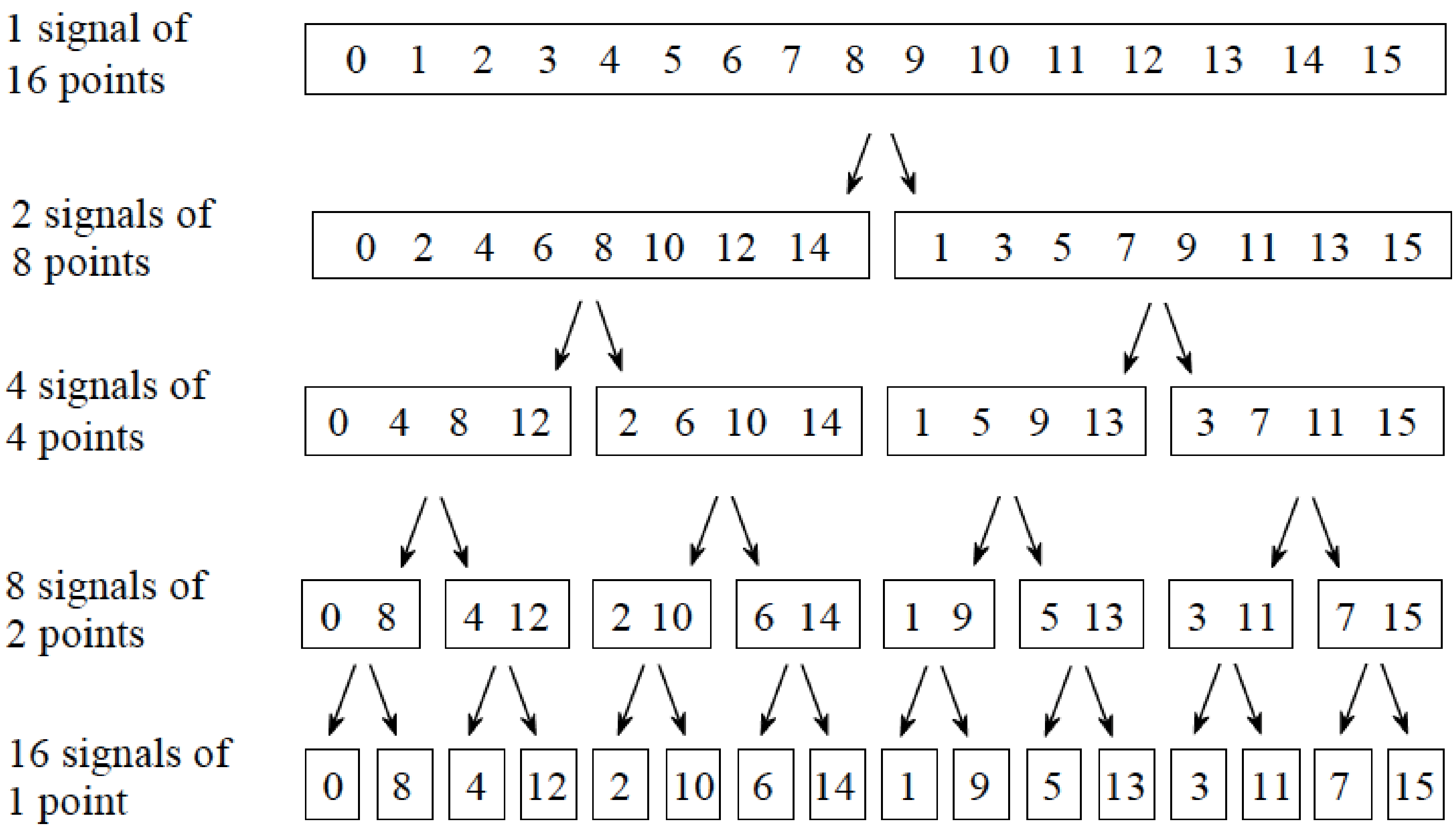
FFT
$$m(n) = 2^{n-1},$$
$$a(n) = 2^{n}$$

Brute force
$$m_{bf}(n) = 2^{2(n-1)},$$
$$a_{bf}(n) = 2^{2n-1}$$

$$c(n) = \frac{m_{bf}(n)}{m(n)} = \frac{a_{bf}(n)}{a(n)} = \frac{2^{n-1}}{n}$$

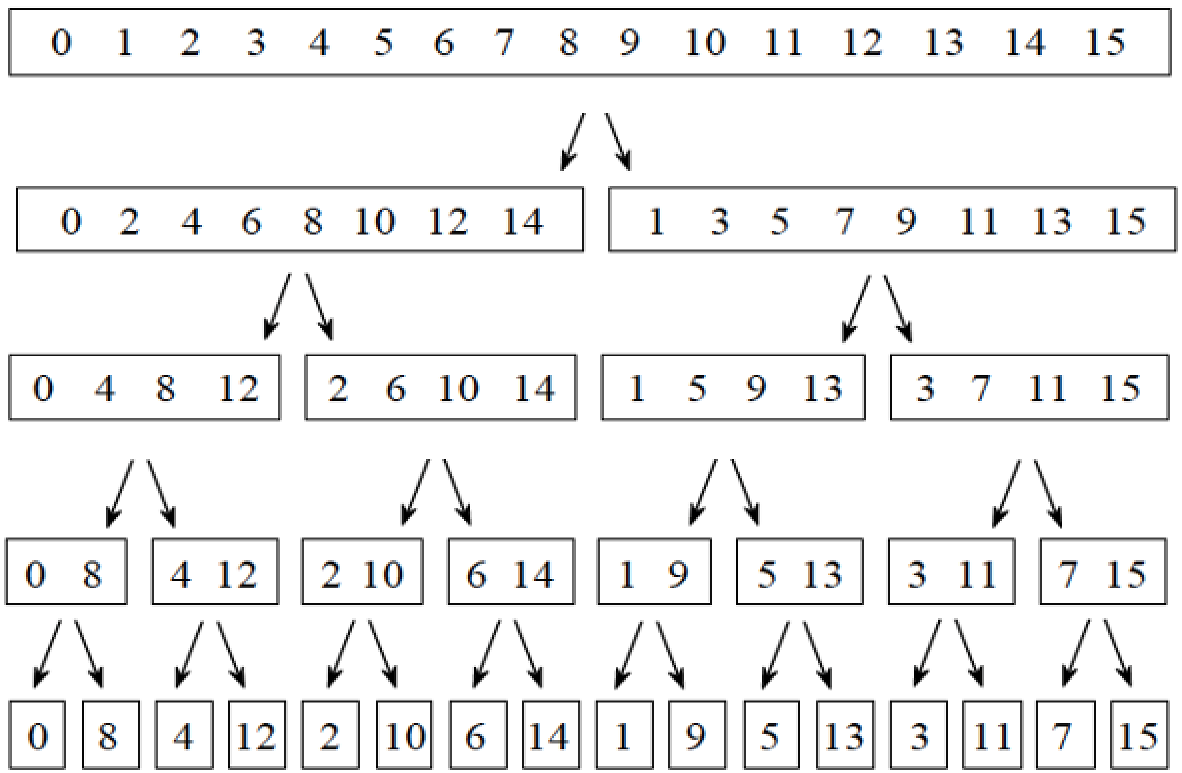


Can we speed up FFT computation further??



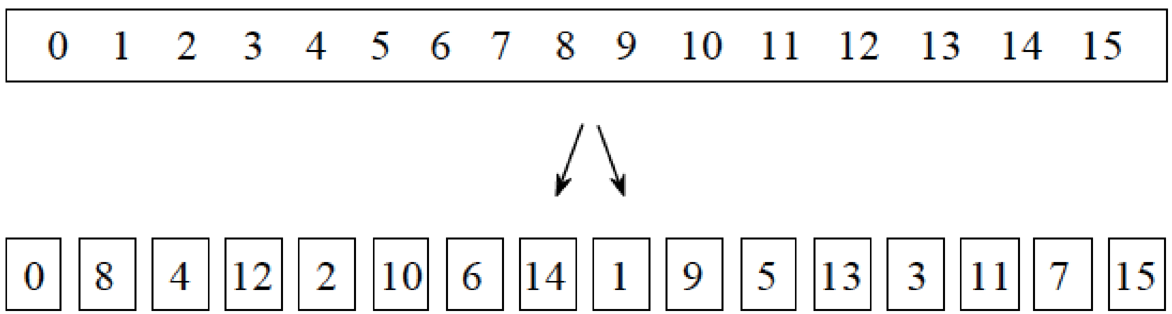
Can we speed up FFT computation further??

- **Current Butterfly implementation:**
  - Dividing array in each stage takes **$\log_2(N)$** steps, but elements need to be reordered.
  - In-place access : more operations and vastly complicated implementation.



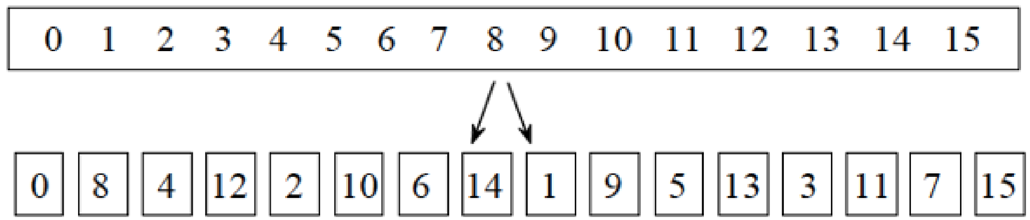
Can we speed up FFT computation further??

- Can we pre-process the array before running FFT ??



Can we speed up FFT computation further??

FFT Pre-processing



Decimal	*Binary*	*Decimal*	*Binary*
0	0000	0	0000
1	0001	8	1000
2	0010	4	0100
3	0011	12	1100
4	0100	2	0010
5	0101	10	1010
6	0110	6	0110
7	0111	14	1110
8	1000	1	0001
9	1001	9	1001
10	1010	5	0101
11	1011	13	1101
12	1100	3	0011
13	1101	11	1011
14	1110	7	0111
15	1111	15	1111

Can we speed up FFT computation further??

FFT Pre-processing

	Original Binary	Pre-processed Binary
	0000	0000
	0001	1000
	0010	0100
	0011	1100
	0100	0010
	0101	1010
	0110	0110
	0111	1110
	1000	0001
	1001	1001
	1010	0101
	1011	1101
	1100	0011
	1101	1011
	1110	0111
	1111	1111

Can we speed up FFT computation further??

Bit Reversal in FFT

- Element exchange is performed with the element in another position as if the bits of the binary index were reversed.

- Perform this preprocess once.

FFT Computation Steps

- Time domain decomposition of the array elements
  - Pre-processed by bit-reversal exchange
- Successively **divide** the array
  - Base case: Element count = 1
- What is the FFT of a single element?
  - The element itself - no new computations necessary at this point (Base case)

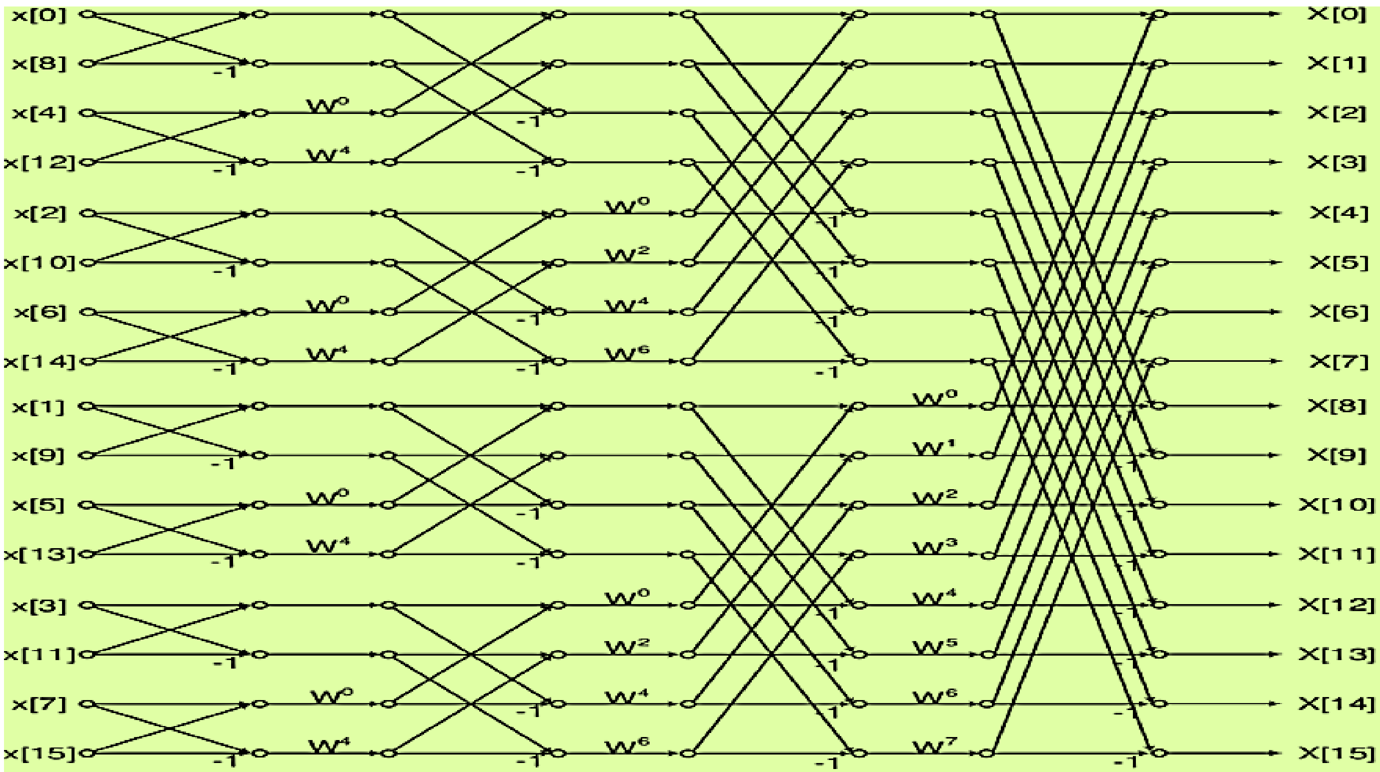
FFT Computation Steps

- At this point, the frequency-domain results look like the following:

0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15

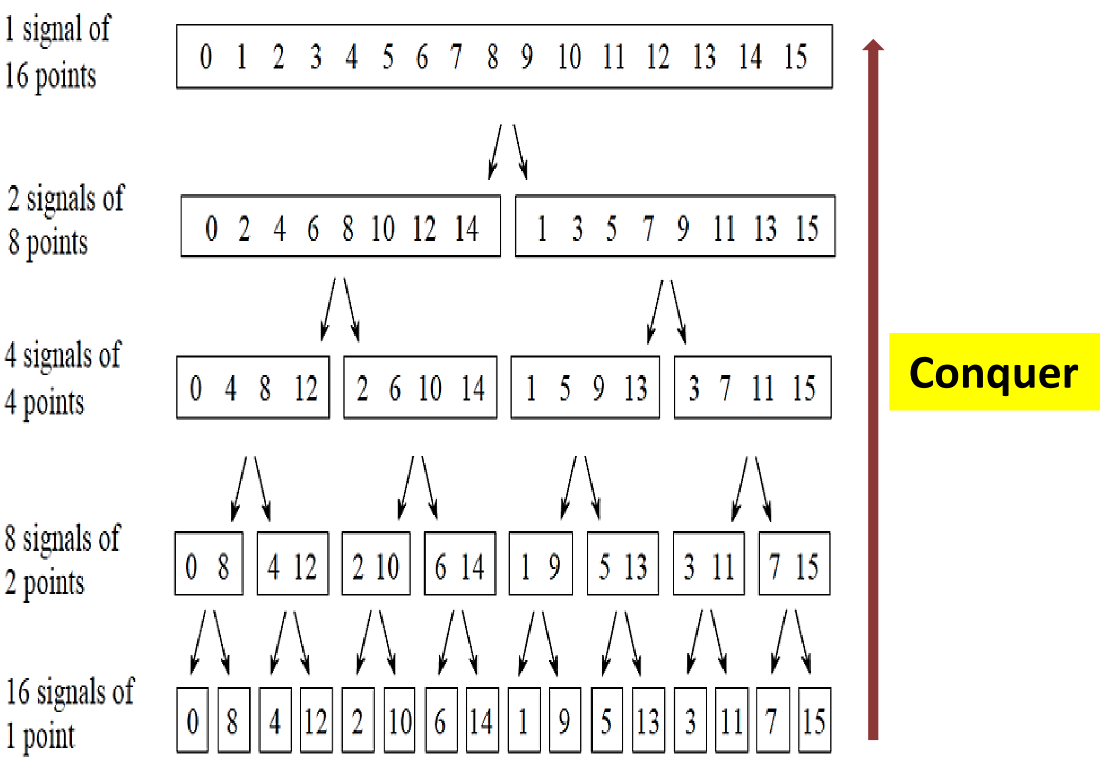
- **Are we done?**

- **NO.** The above result is simply the FFT of each element in array (16 in number)



FFT Computation Steps

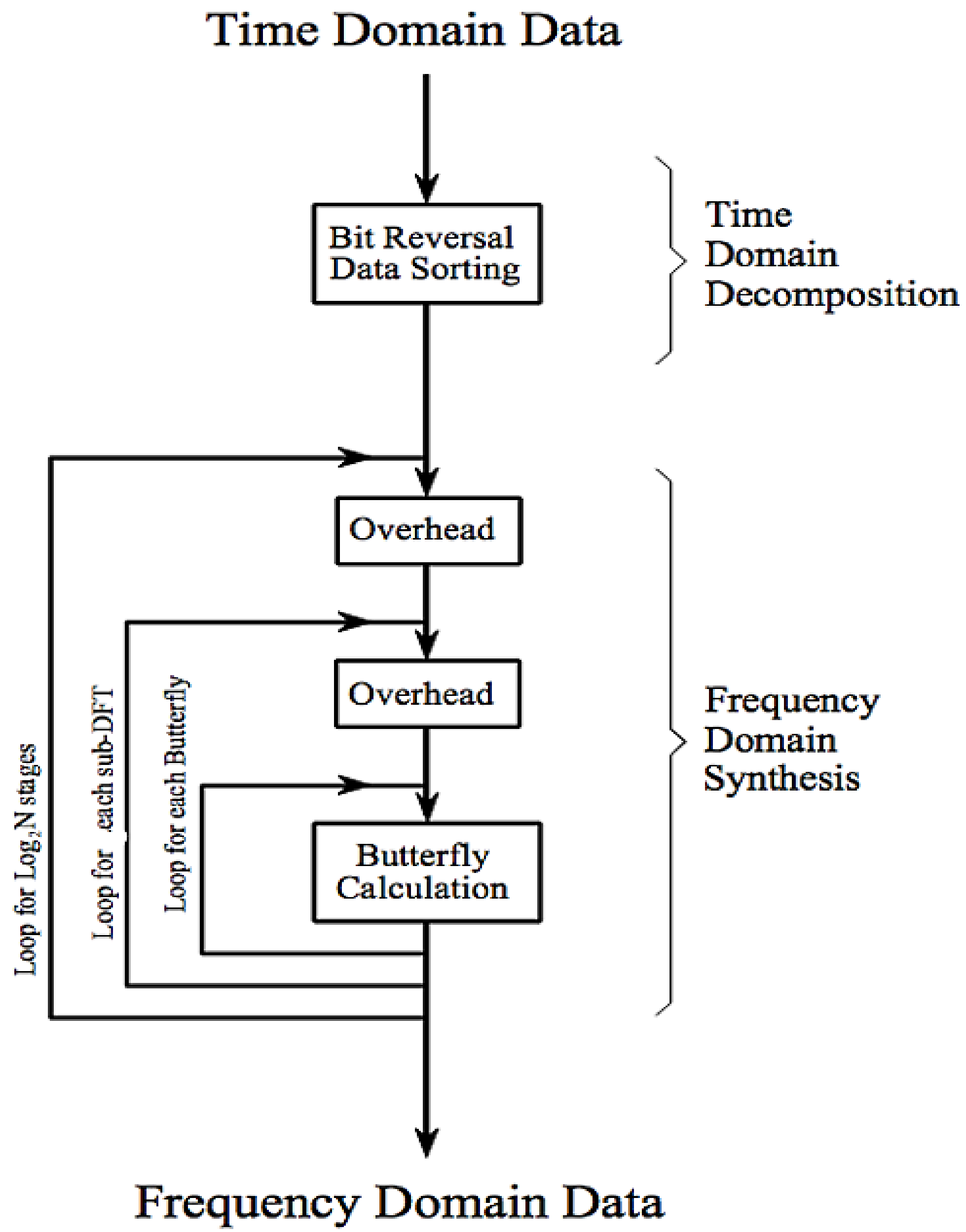
- **Are we done?**
- **NO.** The above result is simply the FFT of each element in array (16 in number)
- Now we use the previously discussed property of even-odd functions to combine:
  - 1-element arrays into 2-element arrays,
  - 2-element arrays into 4-element arrays,
  - 4-element arrays into 8-element arrays,
  - 8-element arrays into **a 16-element result**



FFT Algorithm

- Gather the input data into a buffer of size N **(N is power of 2)**
- Perform bit-reversal exchange operation
- For count = 0 to $\log_2(N)$
  - Apply butterfly stage calculations to elements of size 2^{count}
- The N-element array contains the Fourier Transform of the original elements

FFT Algorithm



Next Lecture

- The image degradation/restoration model

- Noise models
  - Important noise probability density functions
  - Periodic noise
  - Estimating noise parameters

- Restoration using spatial filters
  - Mean filters
  - Order-static filters
  - Adaptive filters