

# ESE 531: Digital Signal Processing

---

Lec 21: April 9, 2019  
Fast Fourier Transform



# Last Time

---

- ❑ Discrete Fourier Transform
  - Linear convolution through circular convolution
    - Overlap and add
    - Overlap and save
  - Circular convolution through DFT
  
- ❑ Today
  - The Fast Fourier Transform



# Circular Convolution

---

□ Circular Convolution:

$$x_1[n] \circledN x_2[n] \triangleq \sum_{m=0}^{N-1} x_1[m] x_2[((n - m))_N]$$

For two signals of length  $N$

**Note: Circular convolution is commutative**

$$x_2[n] \circledN x_1[n] = x_1[n] \circledN x_2[n]$$

# Circular Convolution as Matrix Operation

## □ Circular Convolution

$$h[n] \otimes x[n] = \sum_{m=0}^{N-1} h[((n-m))_N] x[m]$$

$$\begin{aligned} h[n] \circledast x[n] &= \begin{bmatrix} h[0] & h[N-1] & \cdots & h[1] \\ h[1] & h[0] & & h[2] \\ & & \vdots & \\ h[N-1] & h[N-2] & & h[0] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \\ &= H_c x \end{aligned}$$


Circulant matrix

# Circular Convolution as Matrix Operation

## □ Circular Convolution

$$\begin{aligned}
 h[n] \circledast x[n] &= \begin{bmatrix} h[0] & h[N-1] & \cdots & h[1] \\ h[1] & h[0] & & h[2] \\ & & \ddots & \\ h[N-1] & h[N-2] & & h[0] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[N-1] \end{bmatrix} \\
 &= H_c x
 \end{aligned}$$

$$W_N = \begin{pmatrix} W_N^{00} & \cdots & W_N^{0n} & \cdots & W_N^{0(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{k0} & \cdots & W_N^{kn} & \cdots & W_N^{k(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{(N-1)0} & \cdots & W_N^{(N-1)n} & \cdots & W_N^{(N-1)(N-1)} \end{pmatrix}$$



# Circular Convolution as Matrix Operation

---

- Diagonalize

$$W_N H_c W_N^{-1} = \begin{bmatrix} H[0] & 0 \dots & 0 \\ 0 & H[1] \dots & 0 \\ \vdots & 0 & H[N-1] \end{bmatrix}$$

# Circular Convolution as Matrix Operation

- Diagonalize

$$W_N H_c W_N^{-1} = \begin{bmatrix} H[0] & 0 \dots & 0 \\ 0 & H[1] \dots & 0 \\ \vdots & 0 & H[N-1] \end{bmatrix}$$

- Right-Multiply by  $W_N$

$$W_N H_c = \begin{bmatrix} H[0] & 0 \dots & 0 \\ 0 & H[1] \dots & 0 \\ \vdots & 0 & H[N-1] \end{bmatrix} W_N$$

# Circular Convolution as Matrix Operation

- Diagonalize

$$W_N H_c W_N^{-1} = \begin{bmatrix} H[0] & 0 \dots & 0 \\ 0 & H[1] \dots & 0 \\ \vdots & 0 & H[N-1] \end{bmatrix}$$

- Right-Multiply by  $W_N$

$$W_N H_c = \begin{bmatrix} H[0] & 0 \dots & 0 \\ 0 & H[1] \dots & 0 \\ \vdots & 0 & H[N-1] \end{bmatrix} W_N$$


- Multiply both sides by  $x$

$$W_N H_c x = \begin{bmatrix} H[0] & 0 \dots & 0 \\ 0 & H[1] \dots & 0 \\ \vdots & 0 & H[N-1] \end{bmatrix} W_N x$$



# Fast Fourier Transform (FFT)

---



# Fast Fourier Transform Algorithms

---

- We are interested in efficient computing methods for the DFT and inverse DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, \dots, N-1$$

$$x[n] = \sum_{k=0}^{N-1} X[k] W_N^{-kn}, \quad n = 0, \dots, N-1$$

$$W_N = e^{-j\left(\frac{2\pi}{N}\right)}.$$



## Reminder: Inverse DFT via DFT

---

- Recall that we can use the DFT to compute the inverse DFT:

$$\mathcal{DFT}^{-1}\{X[k]\} = \frac{1}{N} (\mathcal{DFT}\{X^*[k]\})^*$$

- Hence, we can just focus on efficient computation of the DFT.
- Straightforward computation of an N-point DFT (or inverse DFT) requires  $N^2$  complex multiplications.



# Computation Order

---

- ❑ *Fast Fourier transform algorithms* enable computation of an  $N$ -point DFT (or inverse DFT) with the order of just  $N \cdot \log_2 N$  complex multiplications.
  - This can represent a huge reduction in computational load, especially for large  $N$ .

$N$	$N^2$	$N \cdot \log_2 N$	$\frac{N^2}{N \cdot \log_2 N}$
16	256	64	4.0
128	16,384	896	18.3
1,024	1,048,576	10,240	102.4
8,192	67,108,864	106,496	630.2

# Computation Order

- ❑ *Fast Fourier transform algorithms* enable computation of an  $N$ -point DFT (or inverse DFT) with the order of just  $N \cdot \log_2 N$  complex multiplications.
  - This can represent a huge reduction in computational load, especially for large  $N$ .

$N$	$N^2$	$N \cdot \log_2 N$	$\frac{N^2}{N \cdot \log_2 N}$
16	256	64	4.0
128	16,384	896	18.3
1,024	1,048,576	10,240	102.4
8,192	67,108,864	106,496	630.2
$6 \times 10^6$	$36 \times 10^{12}$	$135 \times 10^6$	$2.67 \times 10^5$

\* 6Mp image size



# Eigenfunction Properties

---

- ❑ Most FFT algorithms exploit the following properties of  $W_N^{kn}$ :

- Conjugate Symmetry

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^*$$

- Periodicity in n and k

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n}$$

- Power


$$W_N^2 = W_{N/2}$$



# FFT Algorithms via Decimation

---

- ❑ Most FFT algorithms decompose the computation of a DFT into successively smaller DFT computations.
  - Decimation-in-time algorithms decompose  $x[n]$  into successively smaller subsequences.
  - Decimation-in-frequency algorithms decompose  $X[k]$  into successively smaller subsequences.
  
- ❑ Note: Assume length of  $x[n]$  is power of 2 ( $N = 2^v$ ). If not, zero-pad to closest power of 2.




# Decimation-in-Time FFT

---

- We start with the DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, \dots, N-1$$





# Decimation-in-Time FFT


---

- We start with the DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, \dots, N-1$$

- Separate the sum into even and odd terms:

$$X[k] = \sum_{n \text{ even}} x[n] W_N^{kn} + \sum_{n \text{ odd}} x[n] W_N^{kn}$$



# Decimation-in-Time FFT

---


- We start with the DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}, \quad k = 0, \dots, N-1$$

- Separate the sum into even and odd terms:

$$X[k] = \sum_{n \text{ even}} x[n] W_N^{kn} + \sum_{n \text{ odd}} x[n] W_N^{kn}$$

- These are two DFTs, each with half the number of samples ( $N/2$ )




# Decimation-in-Time FFT

---

$$X[k] = \sum_{n \text{ even}} x[n] W_N^{kn} + \sum_{n \text{ odd}} x[n] W_N^{kn}$$

Let  $n = 2r$  ( $n$  even) and  $n = 2r + 1$  ( $n$  odd):



# Decimation-in-Time FFT

---

$$X[k] = \sum_{n \text{ even}} x[n] W_N^{kn} + \sum_{n \text{ odd}} x[n] W_N^{kn}$$

Let  $n = 2r$  ( $n$  even) and  $n = 2r + 1$  ( $n$  odd):

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{(2r+1)k}$$



# Decimation-in-Time FFT

---

Let  $n = 2r$  ( $n$  even) and  $n = 2r + 1$  ( $n$  odd):

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{2rk} \end{aligned}$$

# Decimation-in-Time FFT

Let  $n = 2r$  ( $n$  even) and  $n = 2r + 1$  ( $n$  odd):

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{2rk} \end{aligned}$$

Note that:

$$W_N^{2rk} = e^{-j\left(\frac{2\pi}{N}\right)(2rk)} = e^{-j\left(\frac{2\pi}{N/2}\right)rk} = W_{N/2}^{rk}$$


Remember this trick, it will turn up often.

# Decimation-in-Time FFT

Let  $n = 2r$  ( $n$  even) and  $n = 2r + 1$  ( $n$  odd):

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{(2r+1)k} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_N^{2rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_N^{2rk} \end{aligned}$$

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk}$$



# Decimation-in-Time FFT

---

Hence:

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk} \\ &\triangleq G[k] + W_N^k H[k], \quad k = 0, \dots, N-1 \end{aligned}$$



# Decimation-in-Time FFT

Hence:

$$\begin{aligned} X[k] &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} + W_N^k \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk} \\ &\triangleq G[k] + W_N^k H[k], \quad k = 0, \dots, N-1 \end{aligned}$$

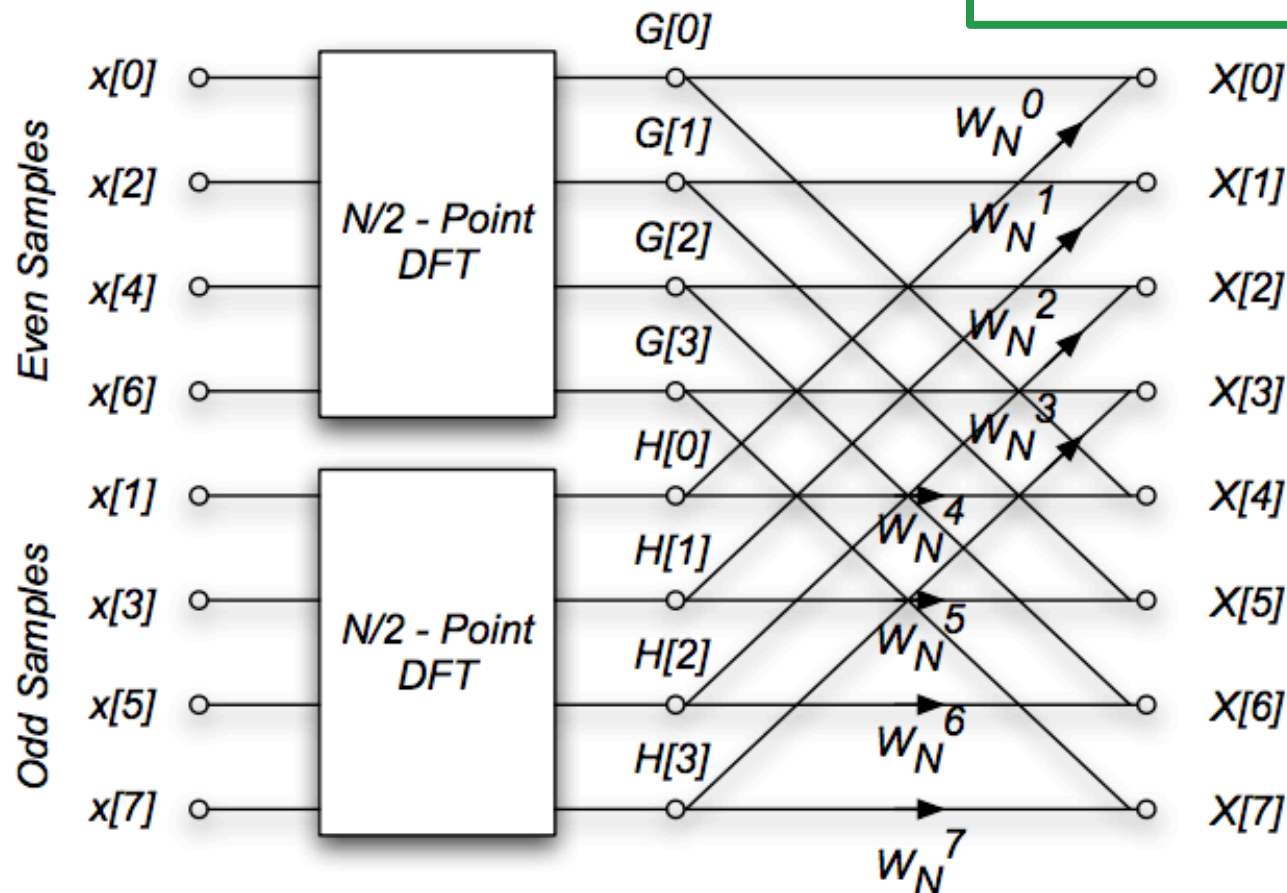
where we have defined:

$$\begin{aligned} G[k] &\triangleq \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} && \Rightarrow \text{DFT of even samples} \\ H[k] &\triangleq \sum_{r=0}^{(N/2)-1} x[2r+1] W_{N/2}^{rk} && \Rightarrow \text{DFT of odd samples} \end{aligned}$$

# Decimation-in-Time FFT

An 8 sample DFT can then be diagrammed as

$$X[k] = G[k] + W_N^k H[k]$$





# Decimation-in-Time FFT

---

Both  $G[k]$  and  $H[k]$  are periodic, with period  $N/2$ . For example

$$G[k] \triangleq \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk}$$

# Decimation-in-Time FFT

Both  $G[k]$  and  $H[k]$  are periodic, with period  $N/2$ . For example

$$\begin{aligned} G[k] &\triangleq \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} \\ G[k + N/2] &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{r(k+N/2)} \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} W_{N/2}^{r(N/2)} = 1 \\ &= \sum_{r=0}^{(N/2)-1} x[2r] W_{N/2}^{rk} \\ &= G[k] \end{aligned}$$

# Decimation-in-Time FFT

□ So,

$$\begin{aligned}G[k + (N/2)] &= G[k] \\H[k + (N/2)] &= H[k]\end{aligned}$$

□ The periodicity of  $G[k]$  and  $H[k]$  allows us to further simplify. For the first  $N/2$  points we calculate  $G[k]$  and  $W_N^k H[k]$ , and then compute the sum

$$X[k] = G[k] + W_N^k H[k] \quad \forall \{k : 0 \leq k < \frac{N}{2}\}.$$

How does periodicity help for  $\frac{N}{2} \leq k < N$ ?



# Decimation-in-Time FFT


---

$$X[k] = G[k] + W_N^k H[k] \quad \forall \{k : 0 \leq k < \frac{N}{2}\}.$$

for  $\frac{N}{2} \leq k < N$ :

$$W_N^{k+(N/2)} = ?$$

$$X[k + (N/2)] = ?$$



# Decimation-in-Time FFT

---

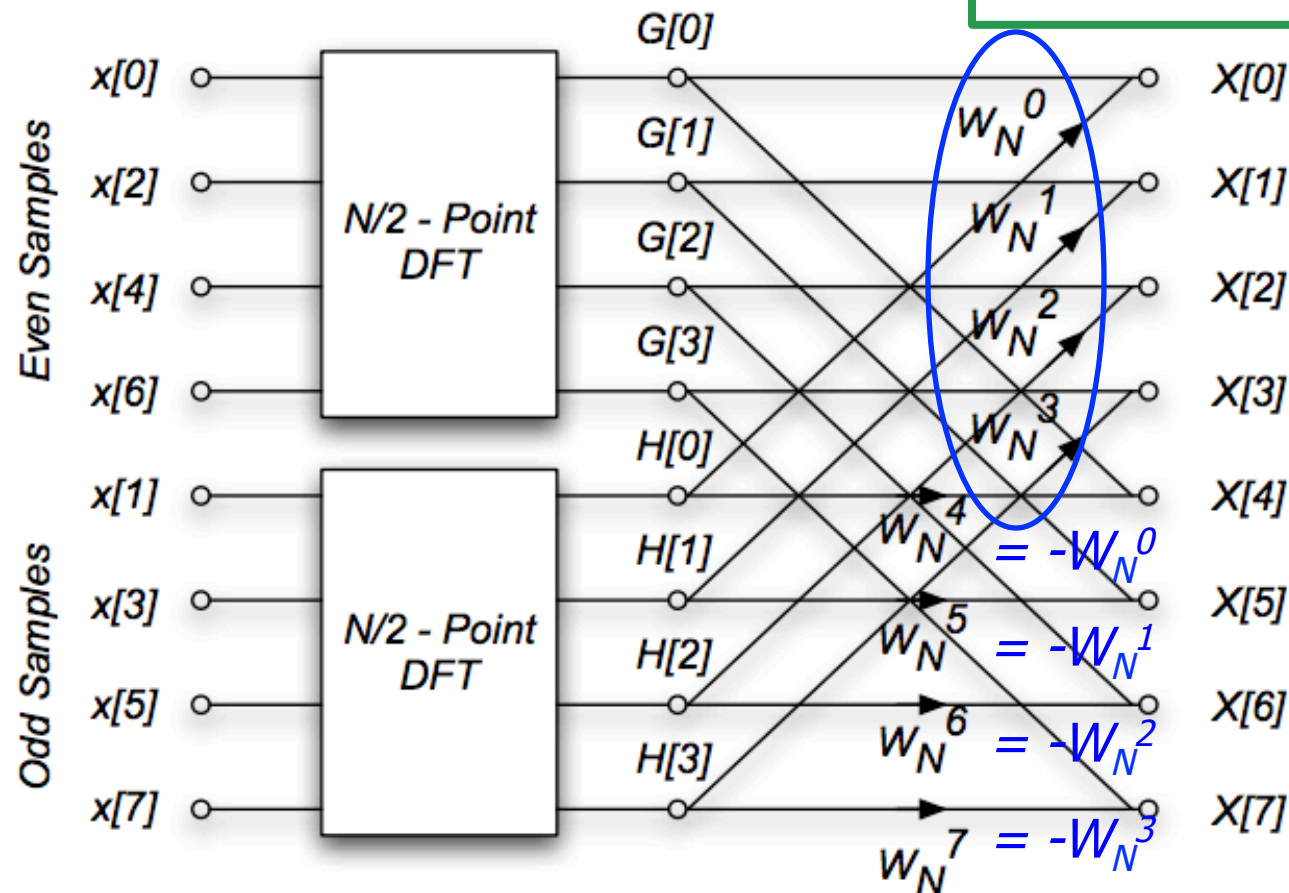
$$X[k + (N/2)] = G[k] - W_N^k H[k]$$

- We previously calculated  $G[k]$  and  $W_N^k H[k]$ .
- Now we only have to compute their difference to obtain the second half of the spectrum. No additional multiplies are required.

# Decimation-in-Time FFT

An 8 sample DFT can then be diagrammed as

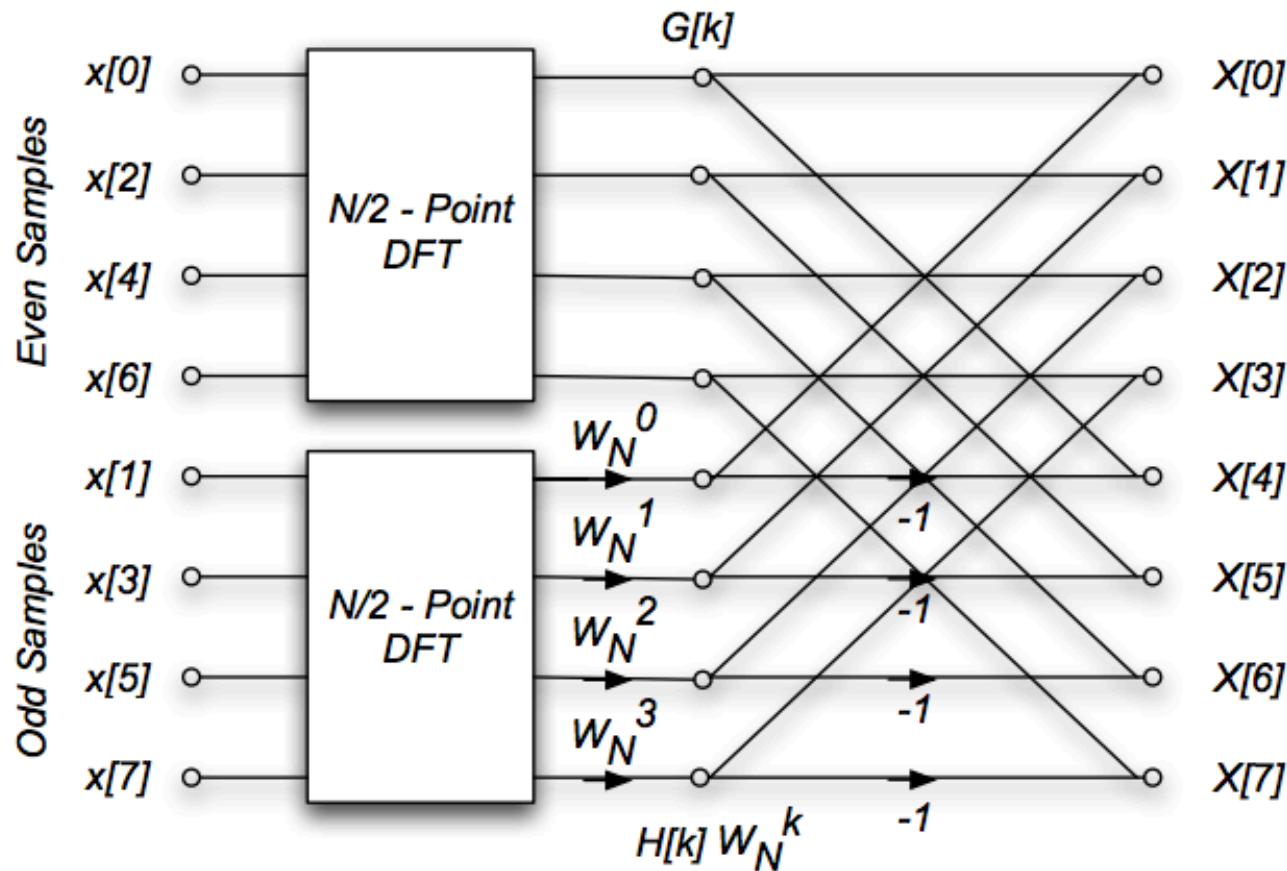
$$X[k] = G[k] + W_N^k H[k]$$





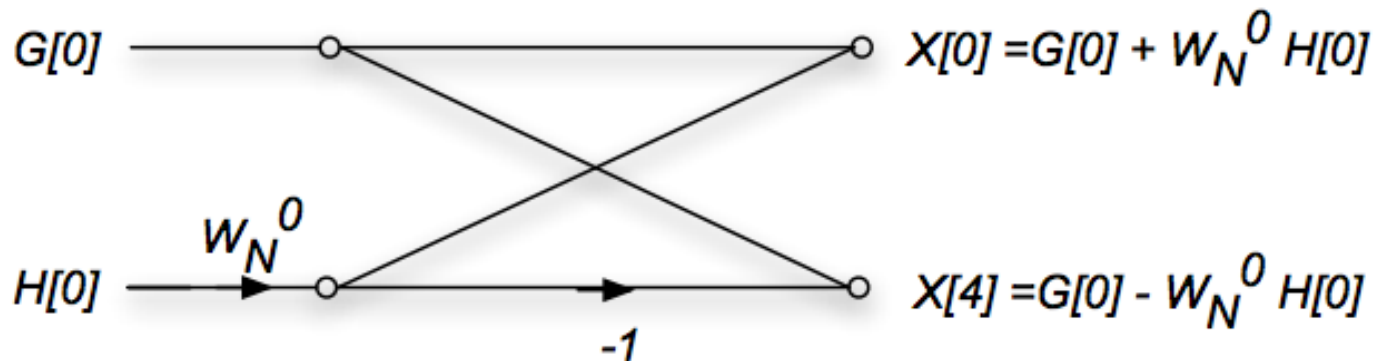
# Decimation-in-Time FFT

The  $N$ -point DFT has been reduced to two  $N/2$ -point DFTs, plus  $N/2$  complex multiplications. The 8 sample DFT is then:



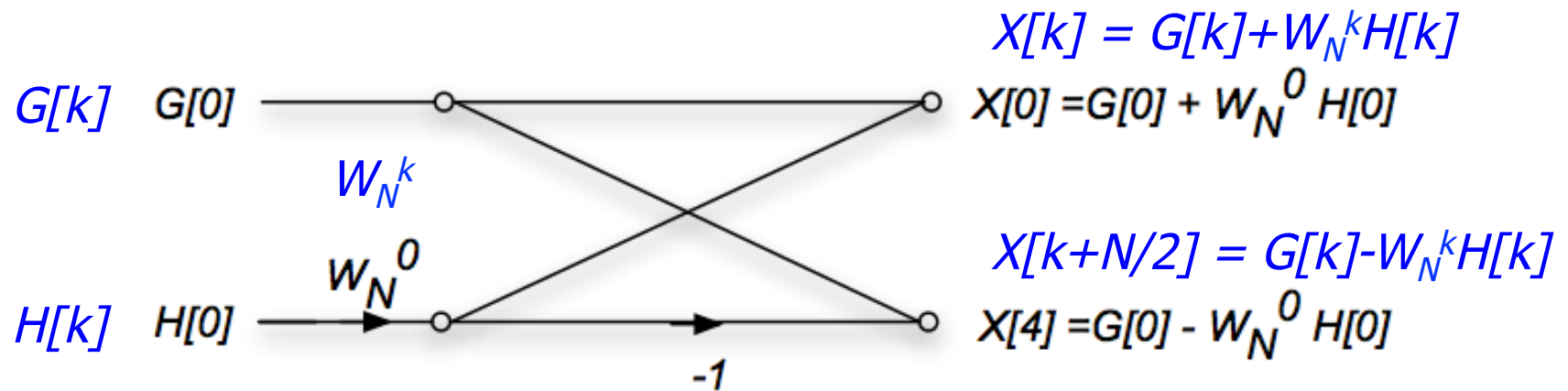
# Decimation-in-Time FFT

- Note that the inputs have been reordered so that the outputs come out in their proper sequence.
- We can define a *butterfly operation*, e.g., the computation of  $X[0]$  and  $X[4]$  from  $G[0]$  and  $H[0]$ :



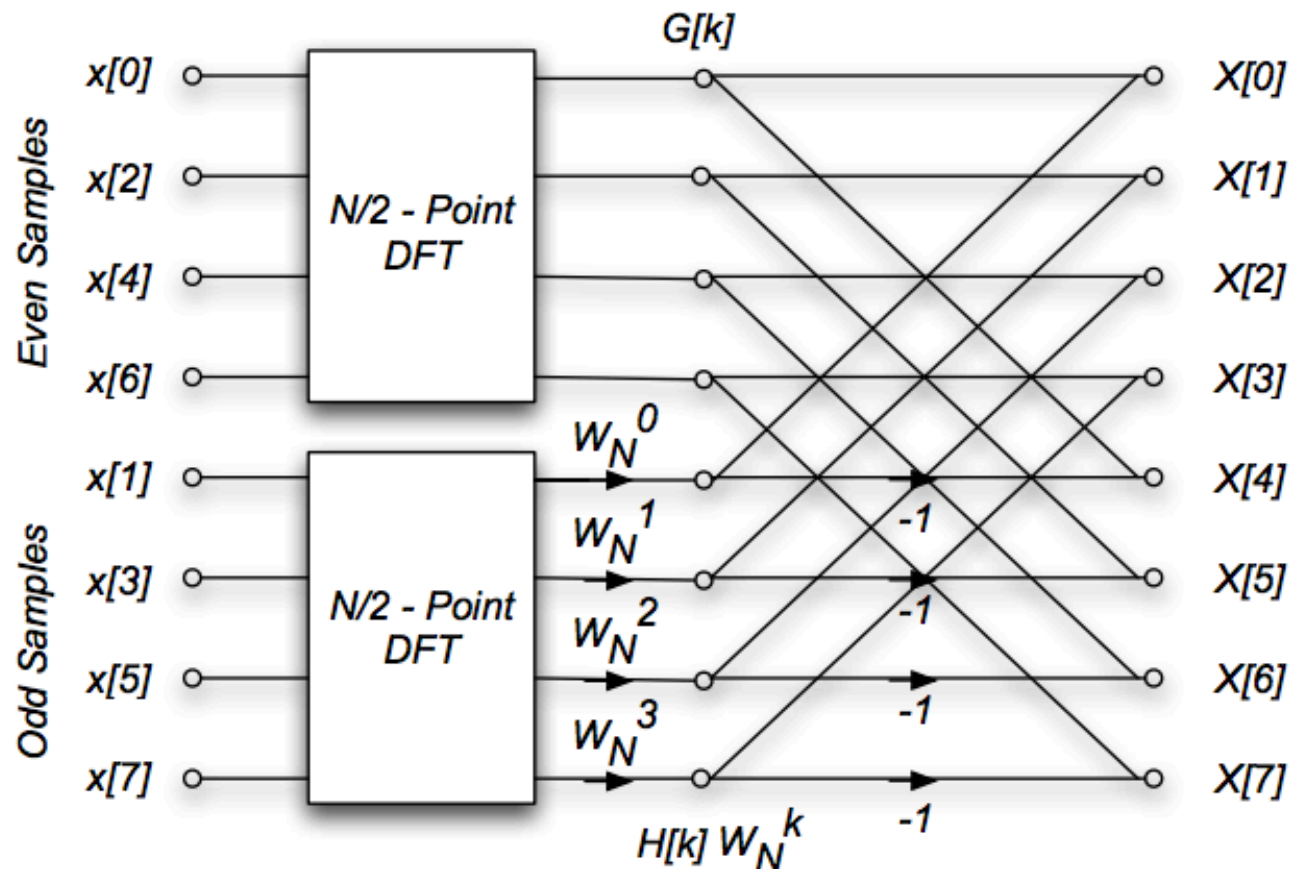
# Decimation-in-Time FFT

- Note that the inputs have been reordered so that the outputs come out in their proper sequence.
- We can define a *butterfly operation*, e.g., the computation of  $X[k]$  and  $X[k+N/2]$  from  $G[k]$  and  $H[k]$ :



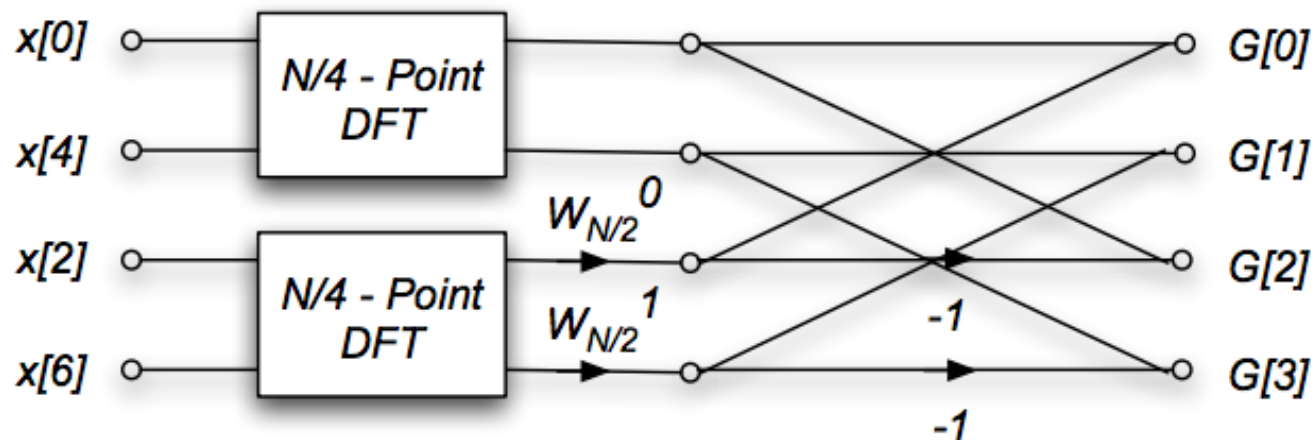
# Decimation-in-Time FFT

- Still  $O(N^2)$  operations.... What should we do?



# Decimation-in-Time FFT

- We can use the same approach for each of the  $N/2$  point DFT's. For the  $N = 8$  case, the  $N/2$  DFTs look like



\*Note that the inputs have been reordered again.



# Decimation-in-Time FFT

---

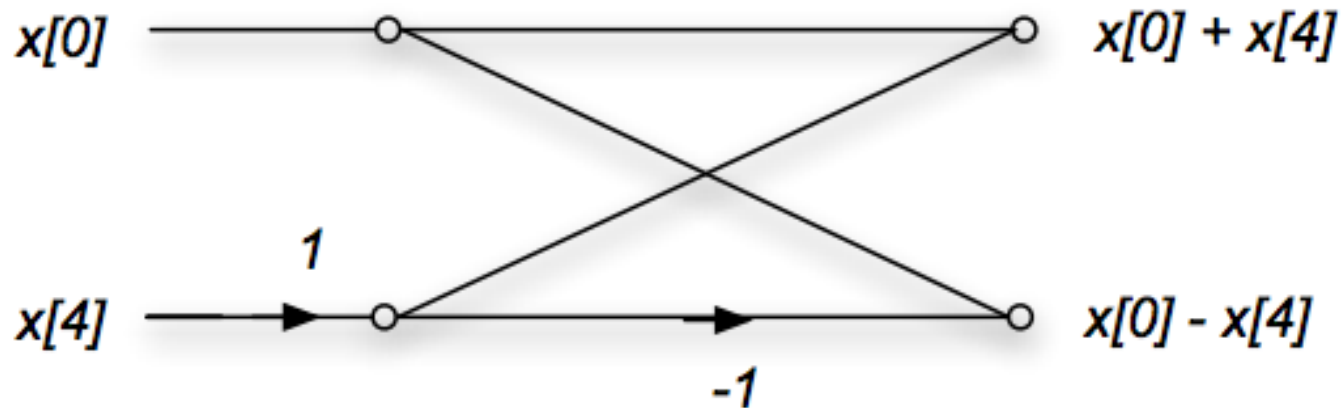
- At this point for the 8 sample DFT, we can replace the  $N/4 = 2$  sample DFT's with a single butterfly. The coefficient is

# Decimation-in-Time FFT

- At this point for the 8 sample DFT, we can replace the  $N/4 = 2$  sample DFT's with a single butterfly. The coefficient is

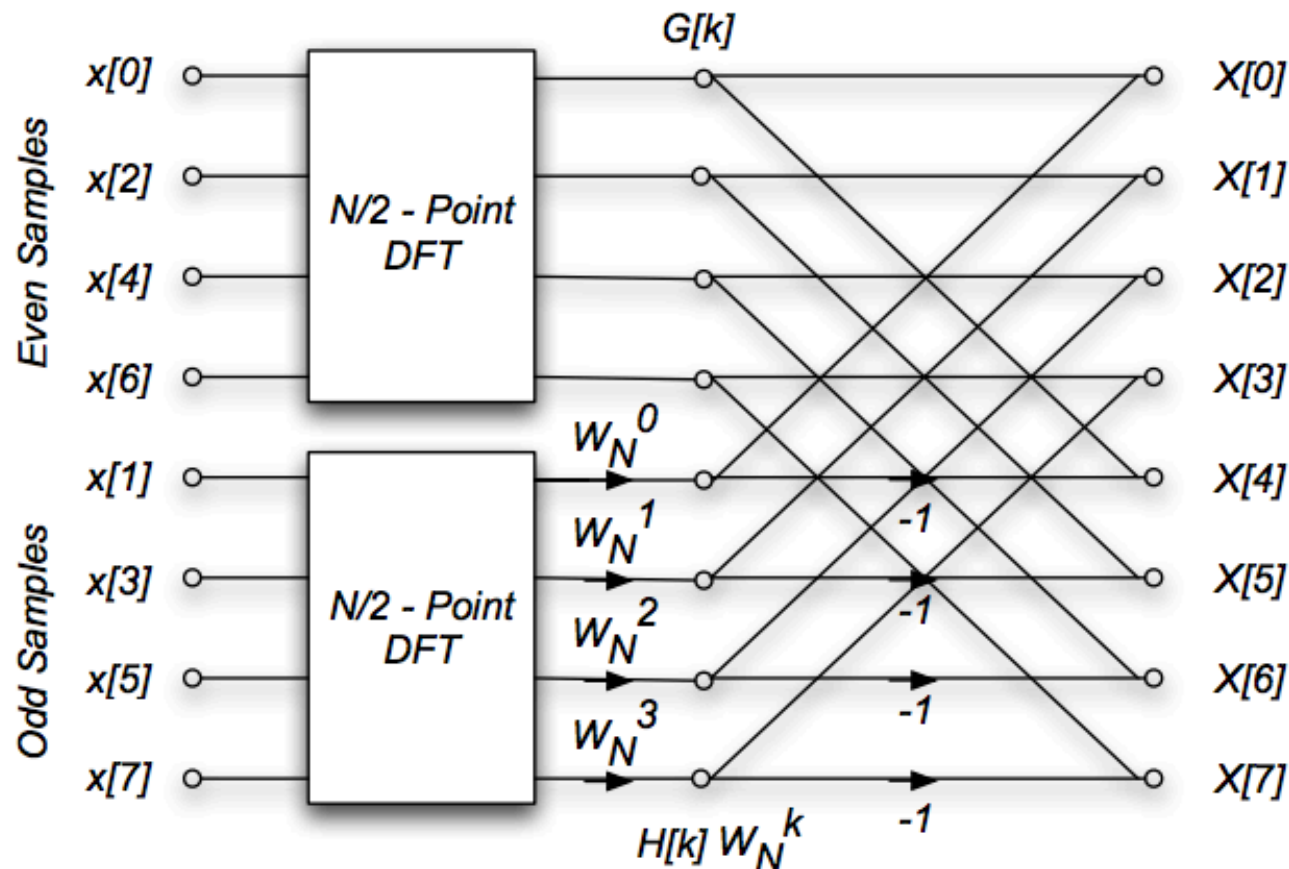
$$W_{N/4} = W_{8/4} = W_2 = e^{-j\pi} = -1$$

The diagram of this stage is then



# Decimation-in-Time FFT

- Replace  $N/2$ -point DFT with 2-point DFT and butterfly operations

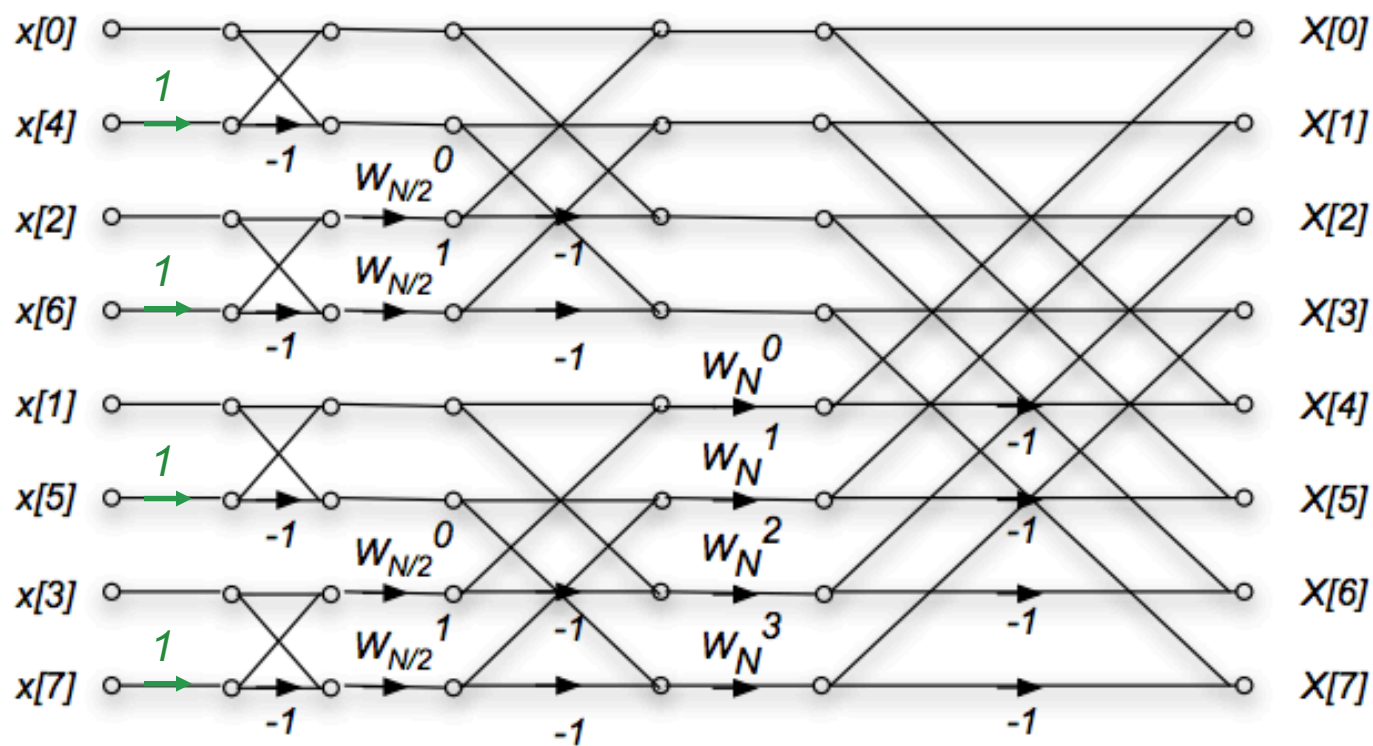






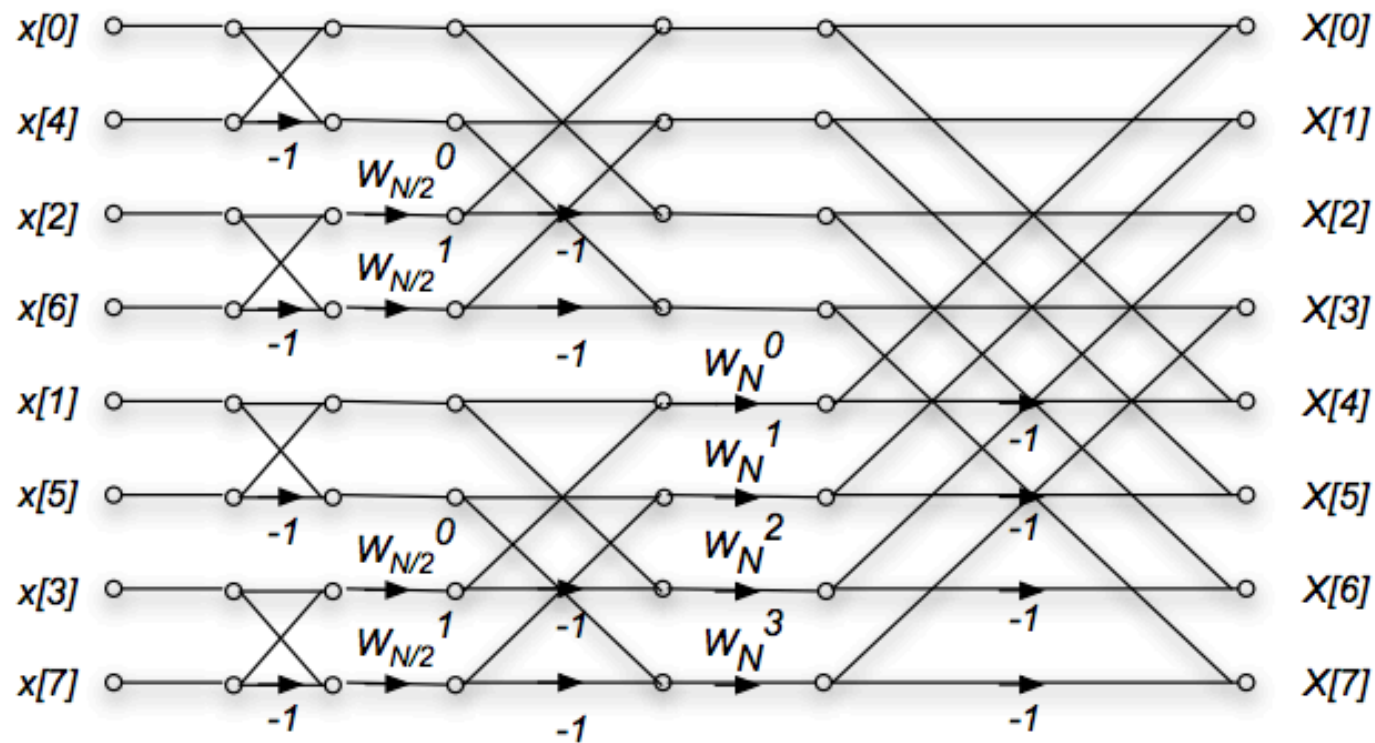
# Decimation-in-Time FFT

Combining all these stages, the diagram for the 8 sample DFT is:




# Decimation-in-Time FFT

Combining all these stages, the diagram for the 8 sample DFT is:




- $3 = \log_2(N) = \log_2(8)$  stages
- $4 = N/2 = 8/2$  multiplications in each stage
  - 1<sup>st</sup> stage has trivial multiplication



# Decimation-in-Time FFT

---


- ❑ In general, there are  $\log_2 N$  stages of decimation-in-time.
- ❑ Each stage requires  $N/2$  complex multiplications, some of which are trivial.
- ❑ The total number of complex multiplications is  $(N/2) \log_2 N$ , or is  $O(N \log_2 N)$



# Decimation-in-Time FFT

---

- ❑ In general, there are  $\log_2 N$  stages of decimation-in-time.
- ❑ Each stage requires  $N/2$  complex multiplications, some of which are trivial.
- ❑ The total number of complex multiplications is  $(N/2) \log_2 N$ , or is  $O(N \log_2 N)$
  
- ❑ The order of the input to the decimation-in-time FFT algorithm must be permuted.
  - First stage: split into odd and even.
    - Zero low-order address bit (LSB) first
  - Next stage repeats with next zero-lower bit
  - Net effect is reversing the bit order of indexes




# Decimation-in-Time FFT

---

This is illustrated in the following table for  $N = 8$ .

Decimal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111




# Decimation-in-Time FFT

---

This is illustrated in the following table for  $N = 8$ .

Decimal	Binary	Bit-Reversed Binary
0	000	000
1	001	100
2	010	010
3	011	110
4	100	001
5	101	101
6	110	011
7	111	111



# Decimation-in-Time FFT

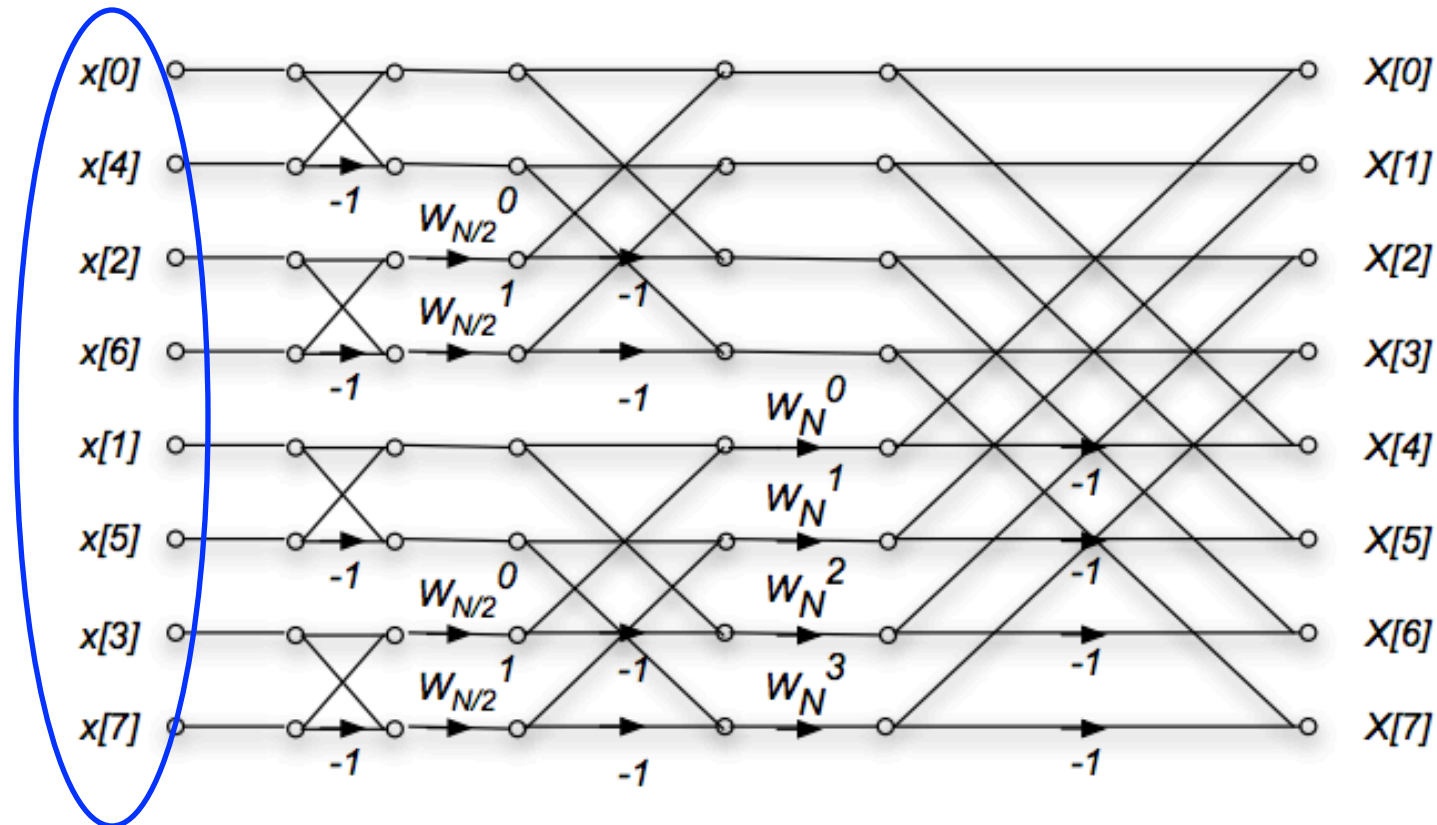
---

This is illustrated in the following table for  $N = 8$ .

Decimal	Binary	Bit-Reversed Binary	Bit-Reversed Decimal
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

# Decimation-in-Time FFT

Combining all these stages, the diagram for the 8 sample DFT is:







# Decimation-in-Frequency FFT

---

The DFT is

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



# Decimation-in-Frequency FFT

---

The DFT is

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

If we only look at the even samples of  $X[k]$ , we can write  $k = 2r$ ,

$$X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)}$$



# Decimation-in-Frequency FFT

---

The DFT is

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

If we only look at the even samples of  $X[k]$ , we can write  $k = 2r$ ,

$$X[2r] = \sum_{n=0}^{N-1} x[n] W_N^{n(2r)}$$

We split this into two sums, one over the first  $N/2$  samples, and the second of the last  $N/2$  samples.

$$X[2r] = \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2r(n+N/2)}$$

# Decimation-in-Frequency FFT

But  $W_N^{2r(n+N/2)} = W_N^{2rn} W_N^{rN} = W_N^{2rn} = W_{N/2}^{rn}$ .

We can then write

$$\begin{aligned} X[2r] &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2r(n+N/2)} \\ &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2rn} \end{aligned}$$

# Decimation-in-Frequency FFT

But  $W_N^{2r(n+N/2)} = W_N^{2rn} W_N^{rN} = W_N^{2rn} = W_{N/2}^{rn}$ .

We can then write

$$\begin{aligned} X[2r] &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2r(n+N/2)} \\ &= \sum_{n=0}^{(N/2)-1} x[n] W_N^{2rn} + \sum_{n=0}^{(N/2)-1} x[n + N/2] W_N^{2rn} \\ &= \sum_{n=0}^{(N/2)-1} (x[n] + x[n + N/2]) W_{N/2}^{rn} \end{aligned}$$

This is the  $N/2$ -length DFT of first and second half of  $x[n]$  summed.



# Decimation-in-Frequency FFT

---

$$\begin{aligned} X[2r] &= \text{DFT}_{\frac{N}{2}} \{(x[n] + x[n + N/2])\} \\ X[2r + 1] &= \text{DFT}_{\frac{N}{2}} \{(x[n] - x[n + N/2]) W_N^n\} \end{aligned}$$

(By a similar argument that gives the odd samples)



# Decimation-in-Frequency FFT

---

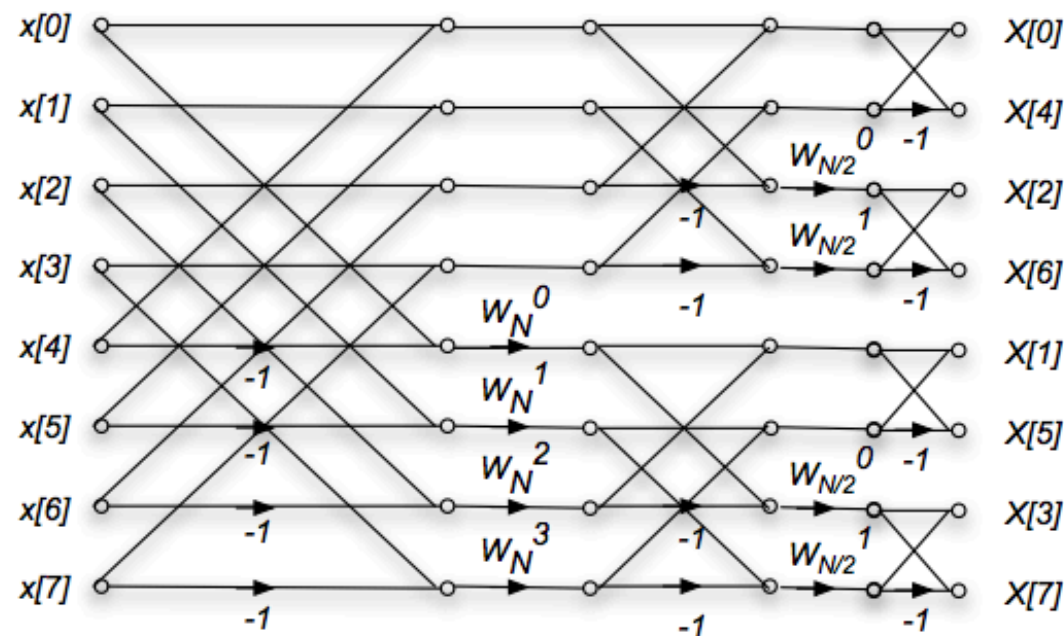
$$\begin{aligned} X[2r] &= \text{DFT}_{\frac{N}{2}} \{(x[n] + x[n + N/2])\} \\ X[2r + 1] &= \text{DFT}_{\frac{N}{2}} \{(x[n] - x[n + N/2]) W_N^n\} \end{aligned}$$

(By a similar argument that gives the odd samples)

- Continue the same approach on the  $N/2$  DFTs, and  $N/4$  DFTs until we reach the 2-point DFT, which is a simple butterfly operation

# Decimation-in-Frequency FFT

The diagram for an 8-point decimation-in-frequency DFT is as follows

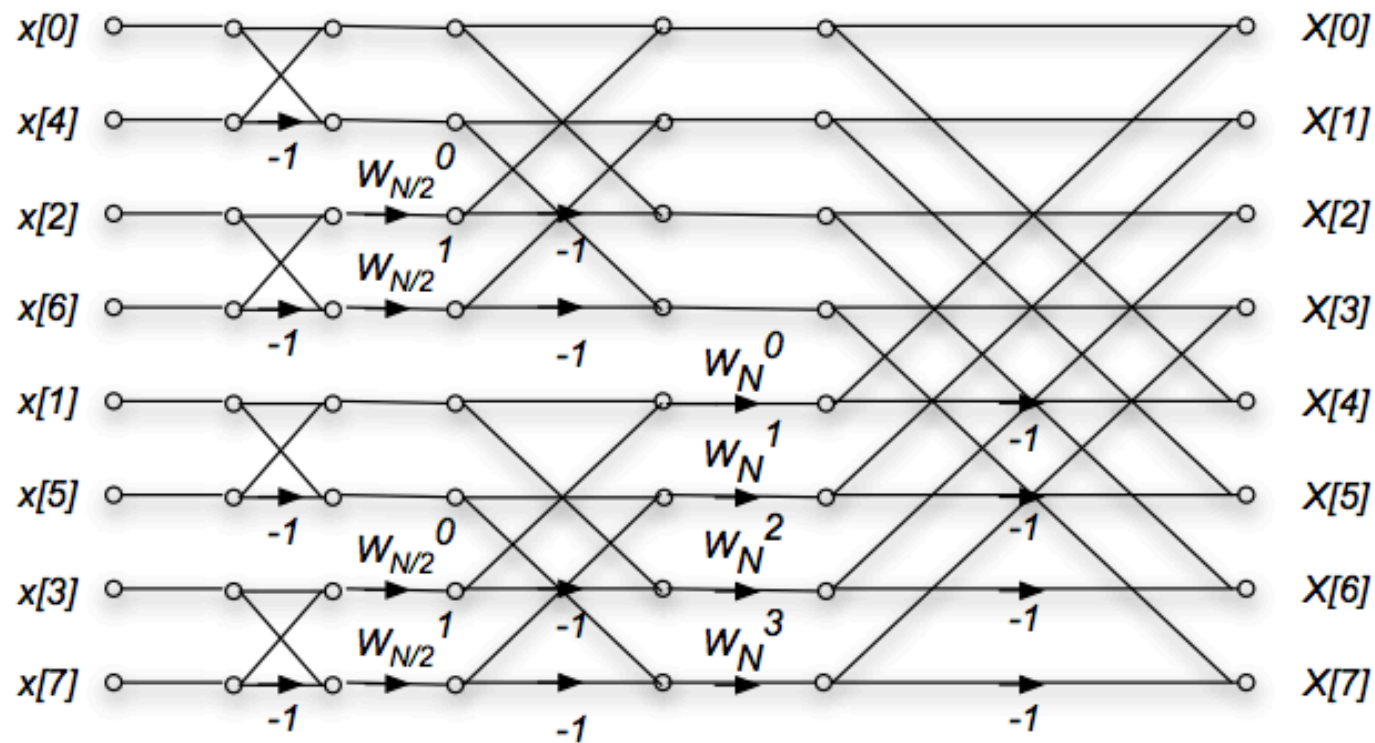


This is just the decimation-in-time algorithm reversed!  
The inputs are in normal order, and the outputs are bit reversed.



# Decimation-in-Time FFT

Combining all these stages, the diagram for the 8 sample DFT is:

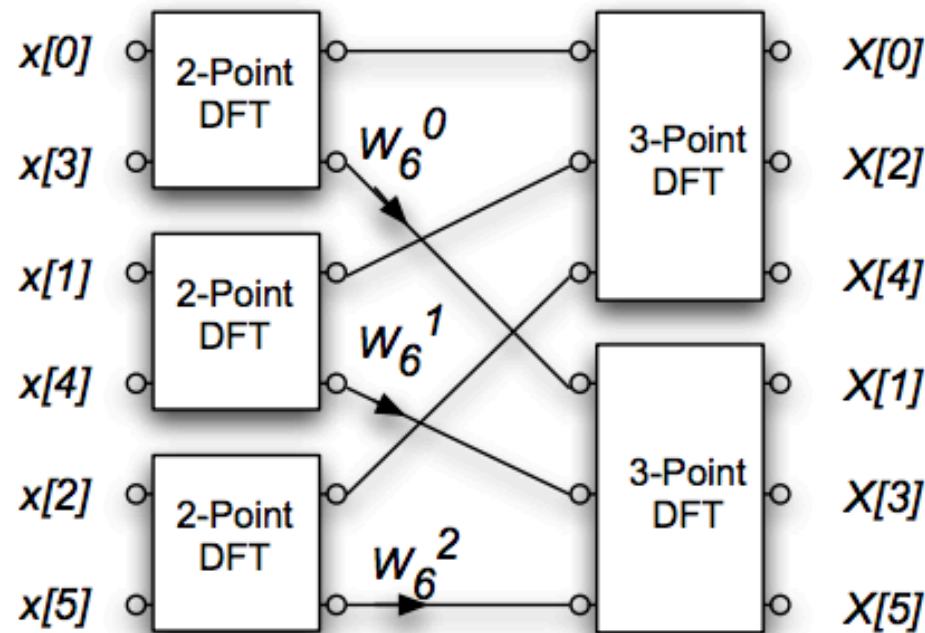


- $3 = \log_2(N) = \log_2(8)$  stages
- $4 = N/2 = 8/2$  multiplications in each stage
  - 1<sup>st</sup> stage has trivial multiplication



## Non-Power-of-2 FFTs

- ❑ A similar argument applies for any length DFT, where the length  $N$  is a composite number
- ❑ For example, if  $N=6$ , a decimation-in-time FFT could compute three 2-point DFTs followed by two 3-point DFTs





# Non-Power-of-2 FFTs

---

- ❑ Good component DFTs are available for lengths up to 20(ish). Many of these exploit the structure for that specific length

- For example, a factor of

$$W_N^{N/4} = e^{-j\frac{2\pi}{N}(N/4)} = e^{-j\frac{\pi}{2}} = -j$$

Just swaps the real and imaginary components of a complex number. Hence a DFT of length 4 doesn't require any complex multiples.

- Half of the multiples of an 8-point DFT also don't require multiplication
  - Composite length FFTs can be very efficient for any length that factors into terms of this order



# Non-Power-of-2 FFTs

---

- ❑ For example  $N = 693$  factors into
  - $N = (7)(9)(11)$
- ❑ each of which can be implemented efficiently. We would perform
  - 9 x 11 DFTs of length 7
  - 7 x 11 DFTs of length 9, and
  - 7 x 9 DFTs of length 11

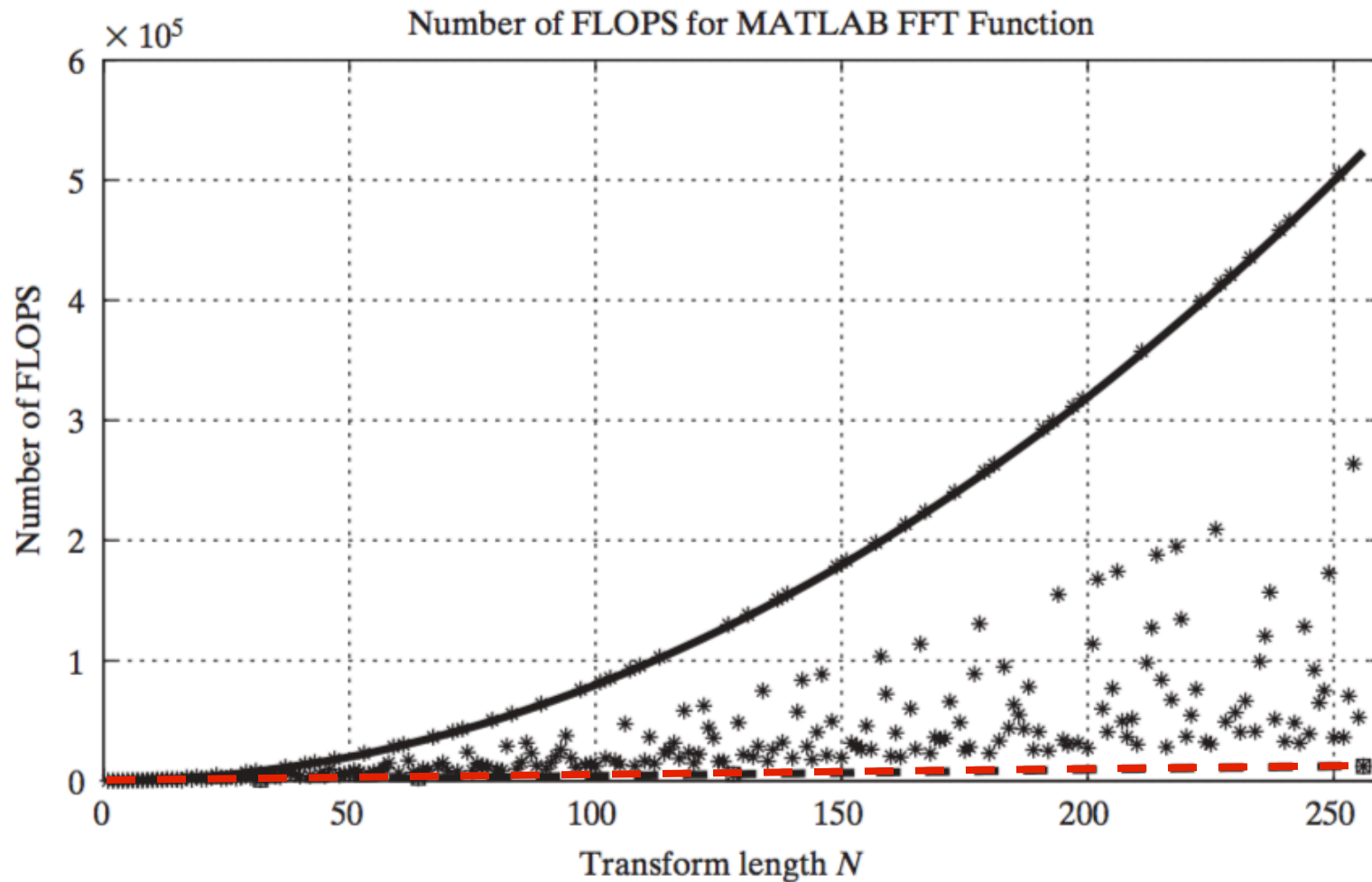


## Non-Power-of-2 FFTs

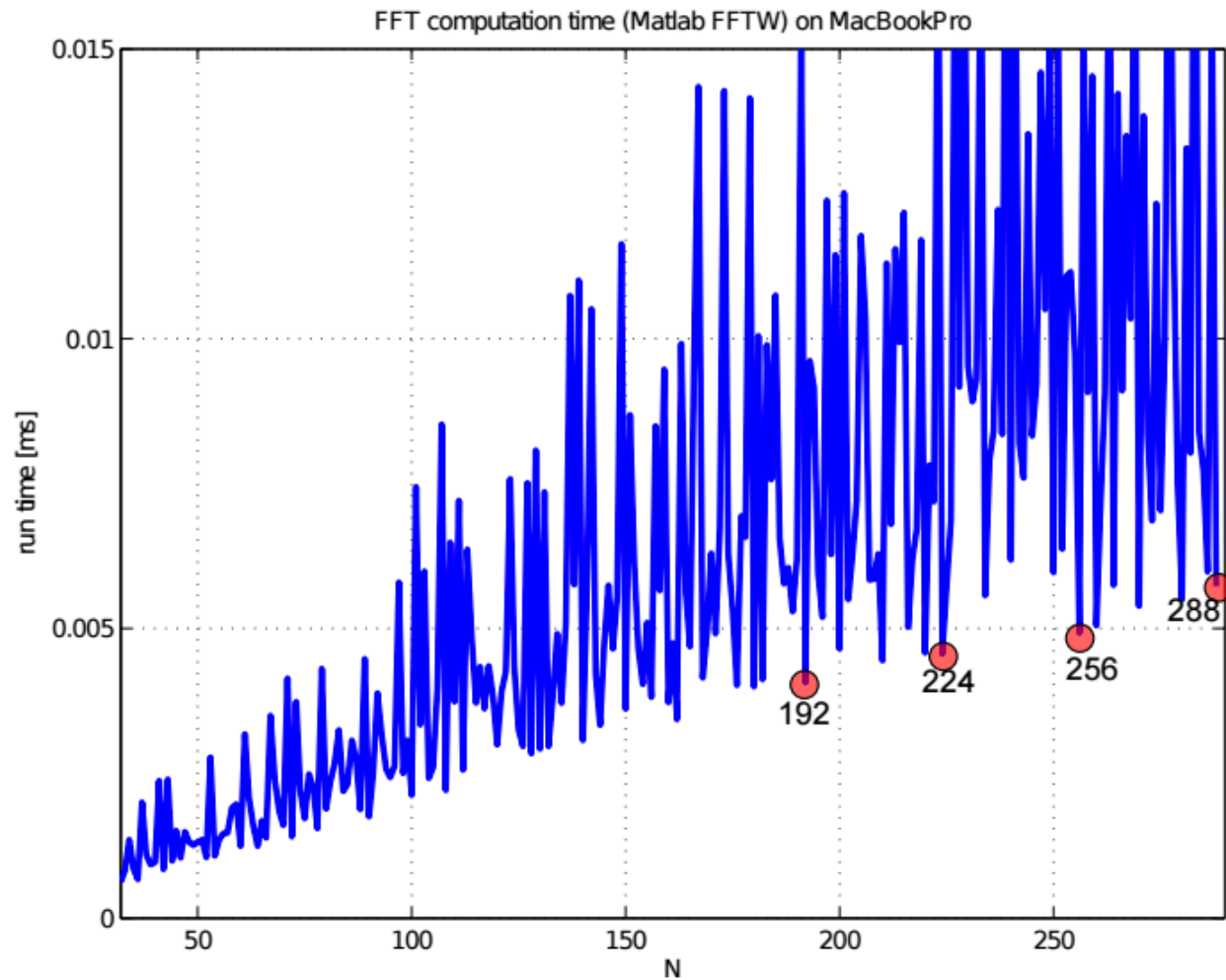
---

- ❑ Historically, the power-of-two FFTs were much faster (better written and implemented).
- ❑ For non-power-of-two length, it was faster to zero pad to power of two.
- ❑ Recently this has changed. The free FFTW package implements very efficient algorithms for almost any filter length. [Matlab has used FFTW since version 6](#)

# FFT Computation FLOPS



# FFT Computation Time



# FFT as Matrix Operation

$$\begin{pmatrix} X[0] \\ \vdots \\ X[k] \\ \vdots \\ X[N-1] \end{pmatrix} = \begin{pmatrix} W_N^{00} & \dots & W_N^{0n} & \dots & W_N^{0(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{k0} & \dots & W_N^{kn} & \dots & W_N^{k(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{(N-1)0} & \dots & W_N^{(N-1)n} & \dots & W_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x[0] \\ \vdots \\ x[n] \\ \vdots \\ x[N-1] \end{pmatrix}$$

- $W_N$  is fully populated  $\rightarrow N^2$  entries



# FFT as Matrix Operation

$$\begin{pmatrix} X[0] \\ \vdots \\ X[k] \\ \vdots \\ X[N-1] \end{pmatrix} = \begin{pmatrix} W_N^{00} & \cdots & W_N^{0n} & \cdots & W_N^{0(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{k0} & \cdots & W_N^{kn} & \cdots & W_N^{k(N-1)} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ W_N^{(N-1)0} & \cdots & W_N^{(N-1)n} & \cdots & W_N^{(N-1)(N-1)} \end{pmatrix} \begin{pmatrix} x[0] \\ \vdots \\ x[n] \\ \vdots \\ x[N-1] \end{pmatrix}$$

- ❑  $W_N$  is fully populated  $\rightarrow N^2$  entries
- ❑ FFT is a decomposition of  $W_N$  into a more sparse form:

$$F_N = \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} W_{N/2} & 0 \\ 0 & W_{N/2} \end{bmatrix} \begin{bmatrix} \text{Even-Odd Perm.} \\ \text{Matrix} \end{bmatrix}$$

- ❑  $I_{N/2}$  is an identity matrix.  $D_{N/2}$  is a diagonal matrix with entries  $1, W_N, \cdots, W_N^{N/2-1}$



# FFT as Matrix Operation

---

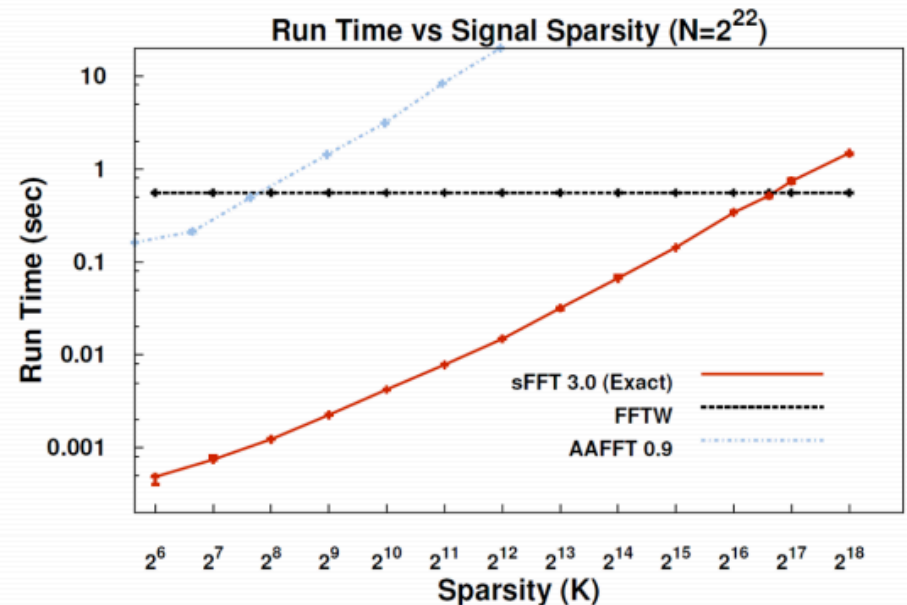
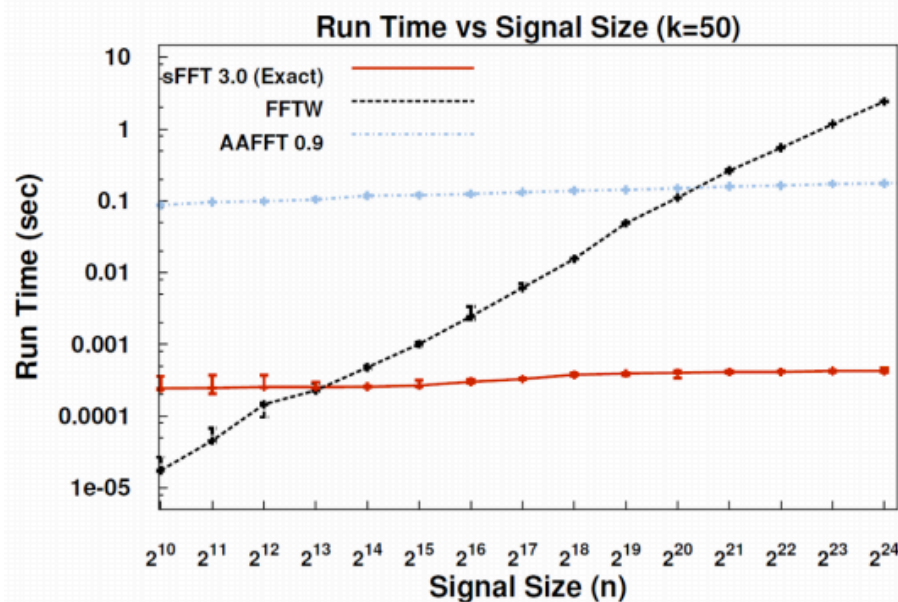
$$F_N = \begin{bmatrix} I_{N/2} & D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} W_{N/2} & 0 \\ 0 & W_{N/2} \end{bmatrix} \begin{bmatrix} \text{Even-Odd Perm.} \\ \text{Matrix} \end{bmatrix}$$

Example:  $N = 4$

$$F_4 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & W_4 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -W_4 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Beyond NlogN

- What if the signal  $x[n]$  has a  $k$  sparse frequency
  - A. Gilbert et. al, “Near-optimal sparse Fourier representations via sampling
  - H. Hassanieh et. al, “Nearly Optimal Sparse Fourier Transform”
  - Others...
  - $O(K \log N)$  instead of  $O(N \log N)$





# Big Ideas

---

## ❑ Fast Fourier Transform

- Enable computation of an  $N$ -point DFT (or  $\text{DFT}^{-1}$ ) with the order of just  $N \cdot \log_2 N$  complex multiplications.
- Most FFT algorithms decompose the computation of a DFT into successively smaller DFT computations.
  - Decimation-in-time algorithms
  - Decimation-in-frequency
- Historically, power-of-2 DFTs had highest efficiency
- Modern computing has led to non-power-of-2 FFTs with high efficiency
- Sparsity leads to reduced computation on order  $K \cdot \log N$



# Admin

---

- ❑ HW 9 due Sunday