EEO 401
Digital Signal Processing

Prof. Mark Fowler

Note Set #25

- FFT Algorithm: Radix-2 Algorithm Development
- Reading Assignment: FFT Write Up Provided

Fast Fourier Transform (FFT) – Efficient Means to Compute DFT

Recall the DFT equation: $$X[k] = \sum_{n=0}^{N-1} \underbrace{x[n]e^{-j2\pi kn/N}}_{\text{1 complex multiply to compute each term… So } N \text{ complex multiplies}} \qquad k = 0,1,2,\ldots,N-1$$

N-1 complex Additions to compute the sum.

1 complex multiply to compute each term…
So N complex multiplies

Must do these N times to compute all N of the DFT points

So… to compute the ***DFT directly*** requires:

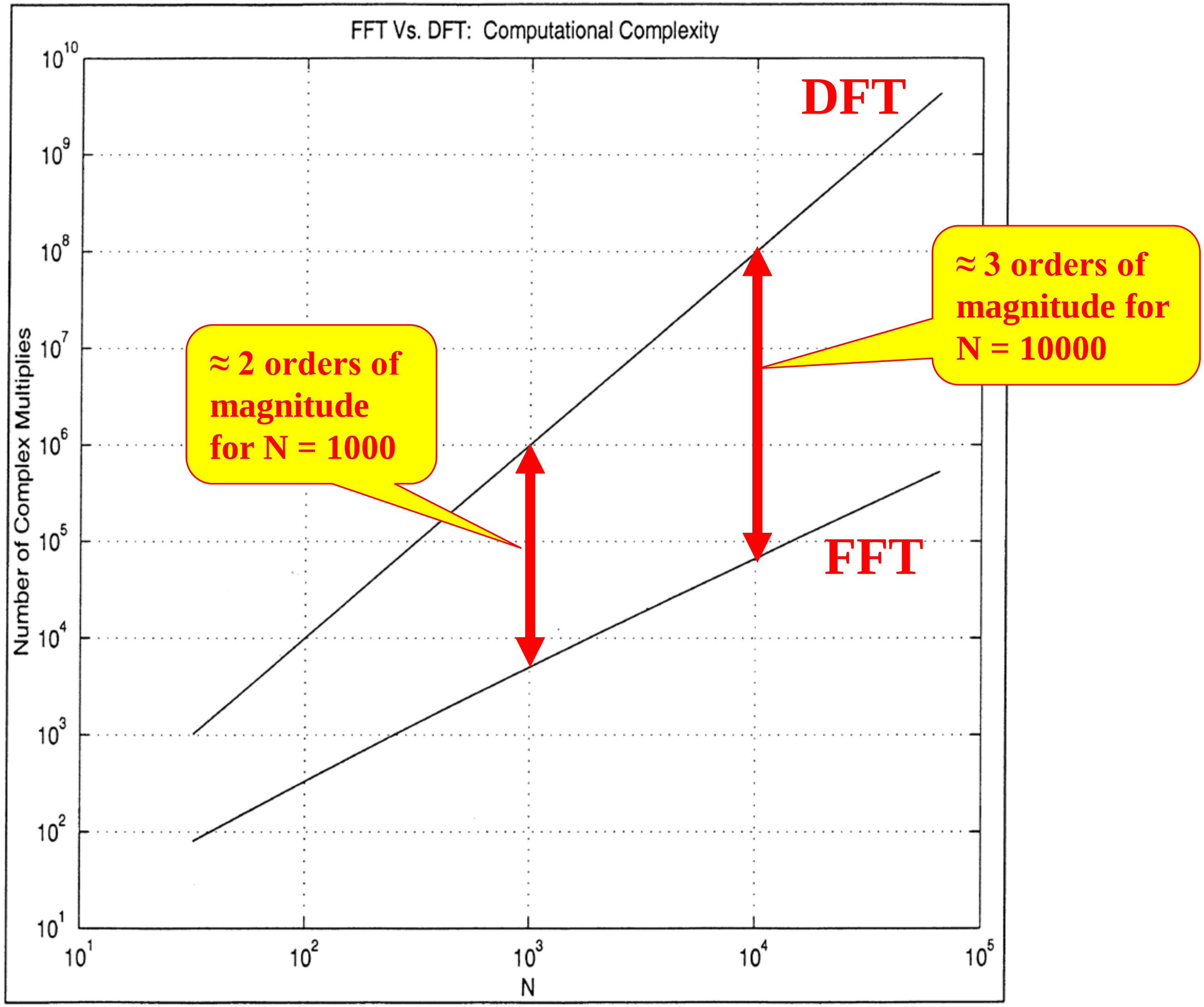
- ***N*²** — N^2 **complex multiplies**
- $N(N\text{-}1) \approx N^2$ **complex additions**

We say this is $O(N^2)$

To compute the DFT using a ***"standard" FFT***:

- $(N/2)\log_2 N$ **complex multiplies**
- $(N/2)\log_2 N$ **complex additions**

We say this is $O(N \log_2 N)$



FFT Vs. DFT: Computational Complexity
DFT
FFT
≈ 2 orders of magnitude for N = 1000
≈ 3 orders of magnitude for N = 10000
Number of Complex Multiplies
N
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
10^10

Motivational Anecdote

The way the FFT computes the DFT efficiently is by exploiting certain inherent structures in the DFT equation that allow efficient computation.

There is a story about Carl Gauss when he was still in primary school that motivates this kind of thing.

His teacher asked the class to add together the numbers from 1 to 100. (Presumably this was to keep the class busy for some time… not sure why he wanted to keep them busy!)

As the other students started what appeared to require lots of work little Gauss thought for a few seconds and then wrote down the answer: 5050.

The trick is to exploit the structure and sum pairs from the outside-in:

1 2 3 4 5 ….. ….. 95 96 97 98 99 100

101

101

101

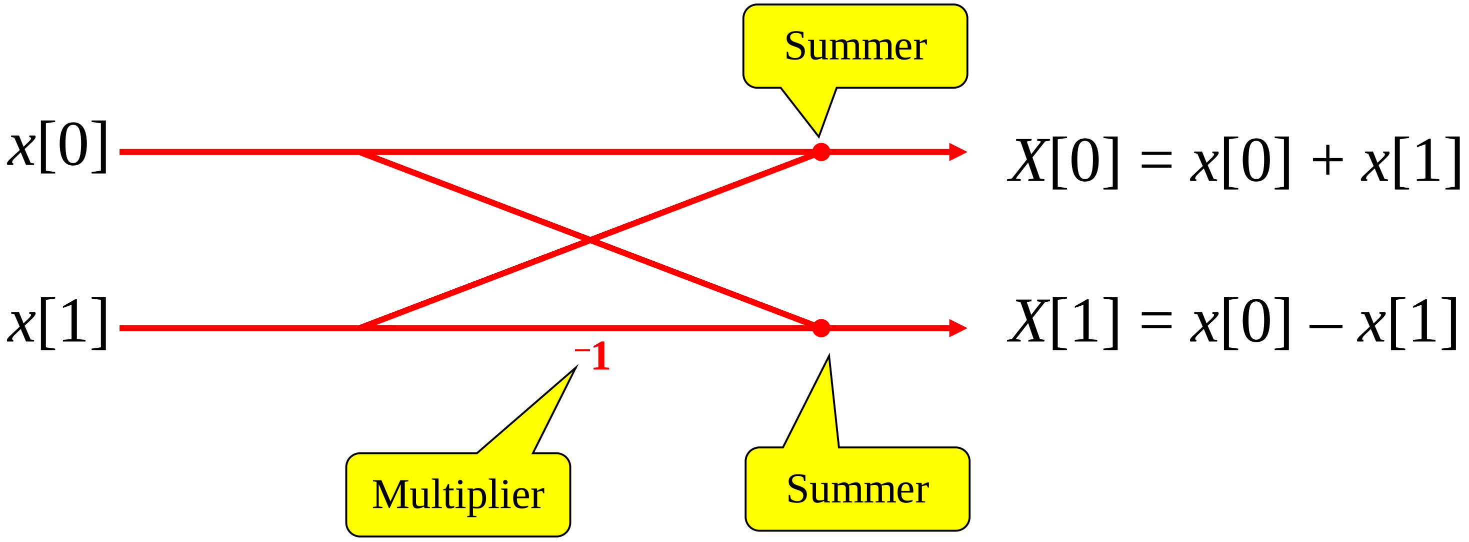
There are 55 such sums so the answer is 55 x 101 = 5050

2-pt DFT Structure

Just as Gauss used 2-pt sums to simplify a 100-pt sum… A key building block in most FFT algorithms is a 2-pt DFT:

$$X[k] = \sum_{n=0}^{1} x[n] e^{-j2\pi kn/2} \qquad k = 0,1$$

$$\boxed{\begin{aligned} X[0] &= x[0]e^{-j\pi 0\times 0} + x[1]e^{-j\pi 0\times 1} = x[0] + x[1] \\ X[1] &= x[0]e^{-j\pi 1\times 0} + x[1]e^{-j\pi 1\times 1} = x[0] - x[1] \end{aligned}}$$



FFT Development (*Radix-2 Decimate-In-Time*)

$N = 2^v$, v is integer

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad k = 0, 1, 2, \ldots, N-1 \qquad W_N = e^{-j2\pi/N}$$

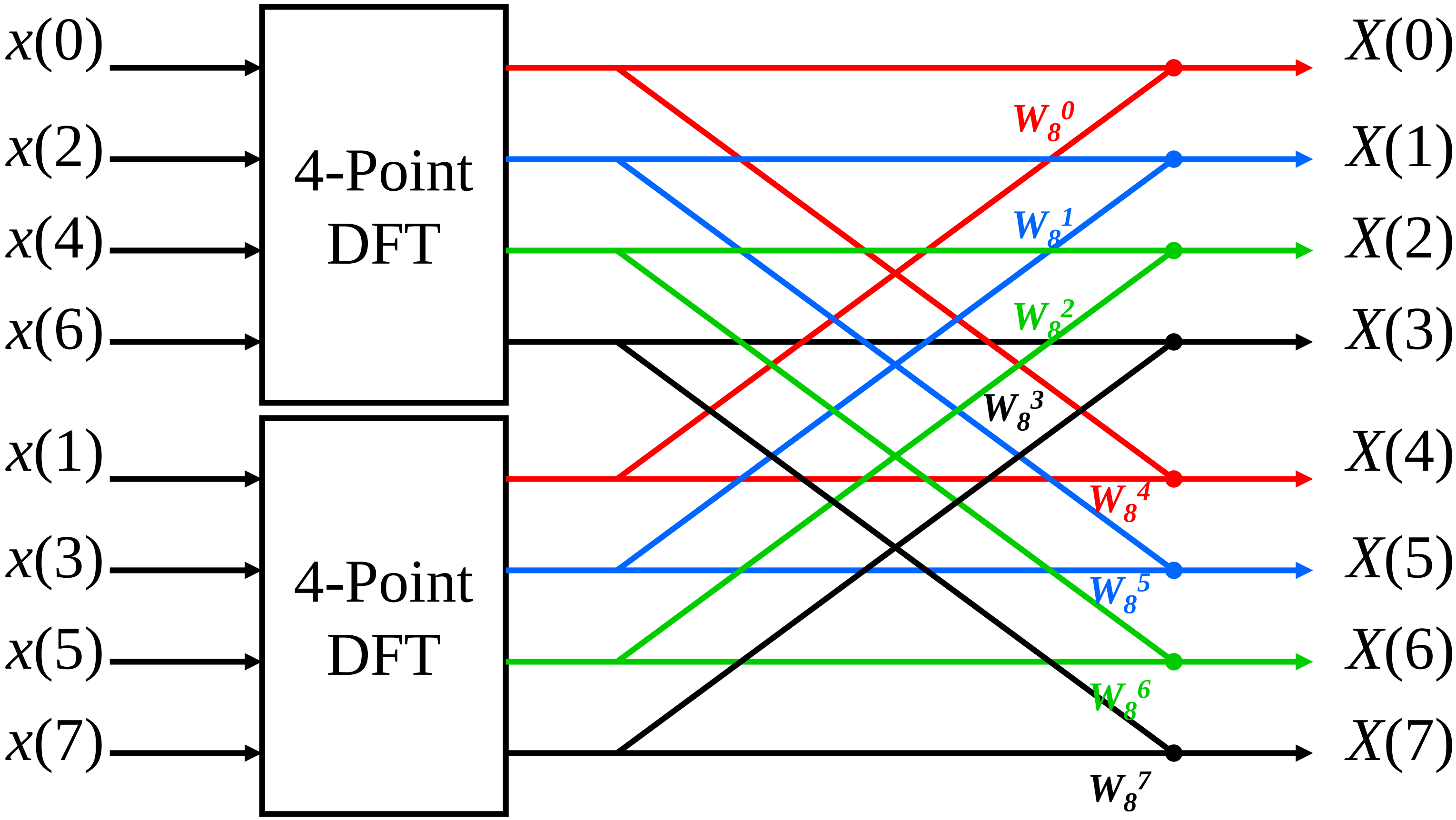
An N–point DFT can be written as the (weighted) sum of two $N/2$–point DFTs (one DFT of the even-indexed samples and one DFT of the odd-indexed samples) as:

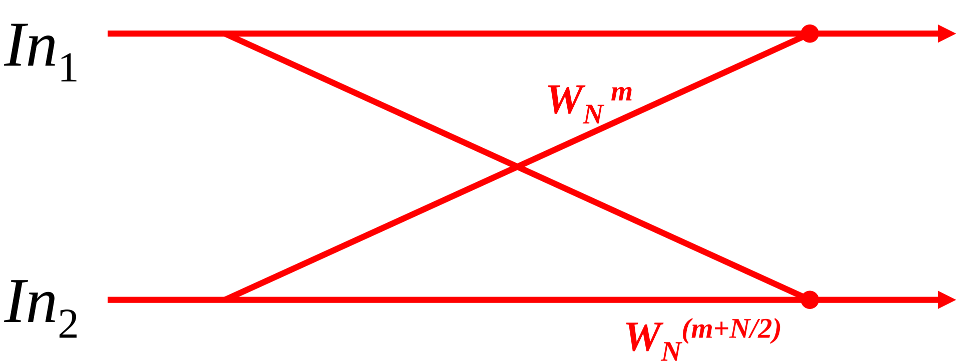
$$X(k) = \sum_{n=0}^{(N/2)-1} x(2n) W_N^{k2n} + \sum_{n=0}^{(N/2)-1} x(2n+1) W_N^{k(2n+1)}$$

$$= \sum_{n=0}^{(N/2)-1} x_e(n) W_N^{k2n} + W_N^k \sum_{n=0}^{(N/2)-1} x_o(n) W_N^{k2n}$$

$$= \underbrace{\sum_{n=0}^{(N/2)-1} x_e(n) W_{N/2}^{kn}}_{N/2\text{-point DFT}} + W_N^k \underbrace{\sum_{n=0}^{(N/2)-1} x_o(n) W_{N/2}^{kn}}_{N/2\text{-point DFT}} \qquad k = 0, 1, 2, \ldots, N-1$$

Each $N/2$-pt DFT really need only be evaluated for k = 0, 1, 2, … , $N/2$ –1, because $(W_{N/2})^{kn}$ is periodic with period $N/2$.

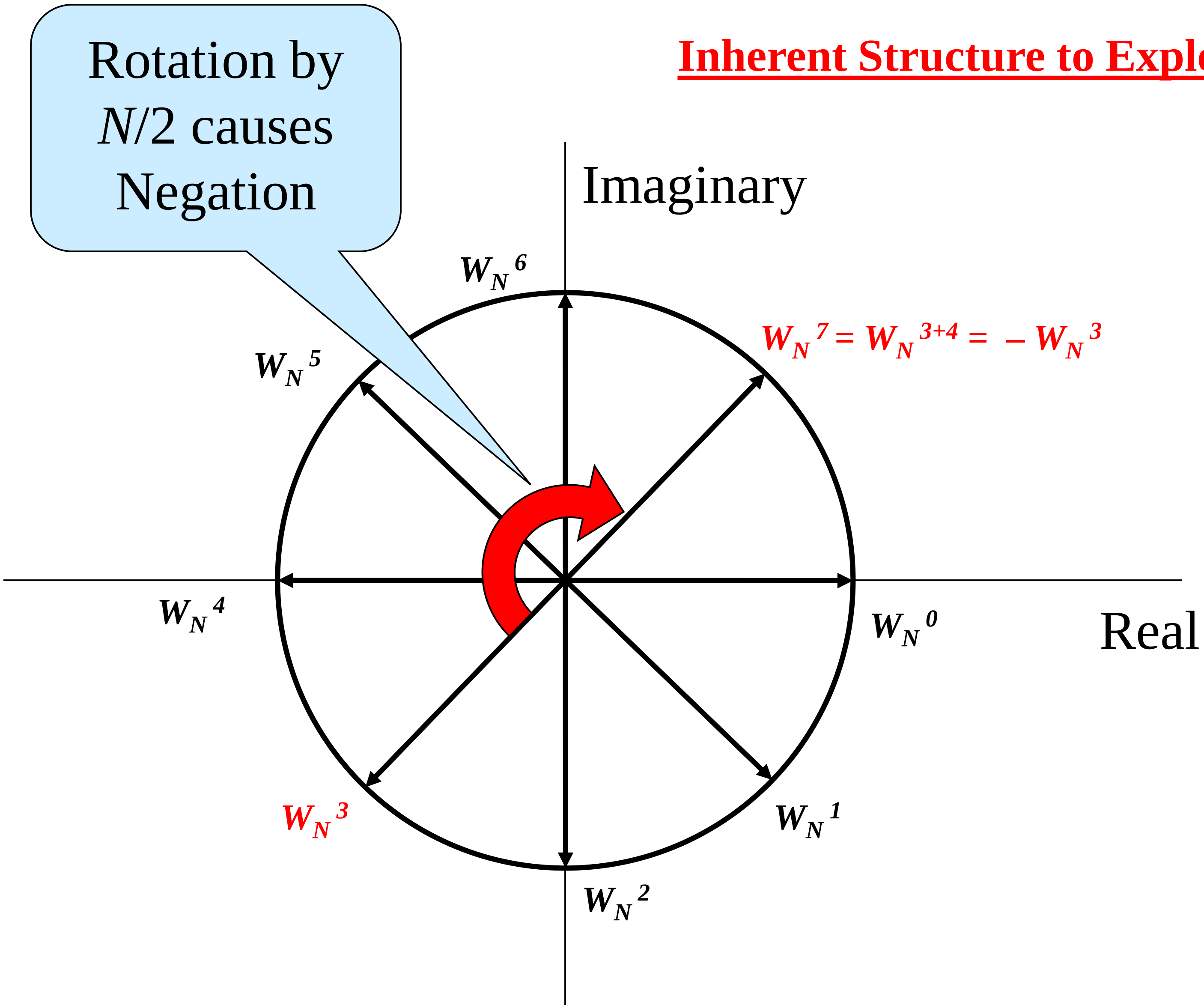




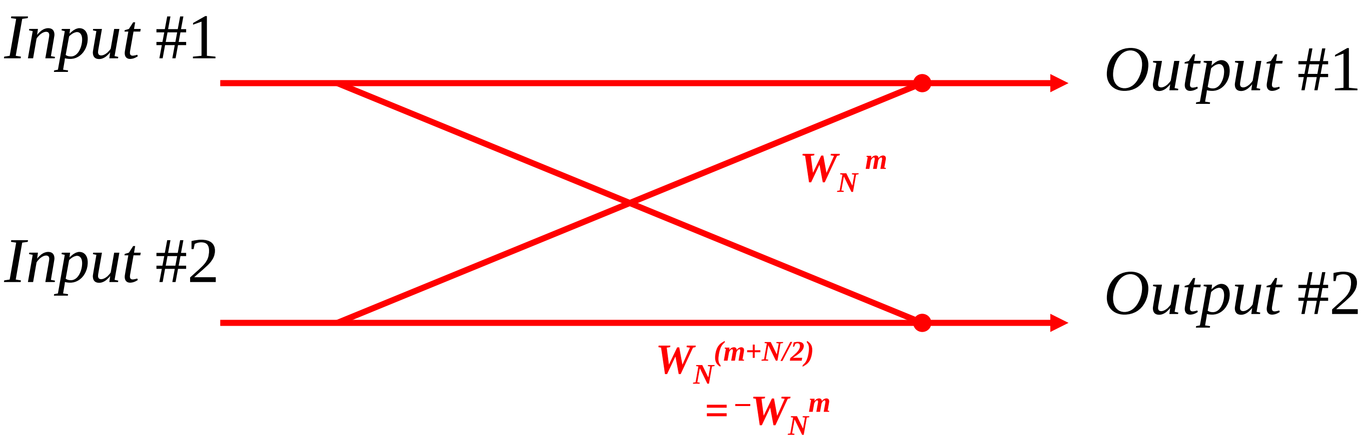
$$Out_1 = In_1 + w_N^{m} In_2$$

$$Out_2 = In_1 + w_N^{(m+N/2)} In_2$$

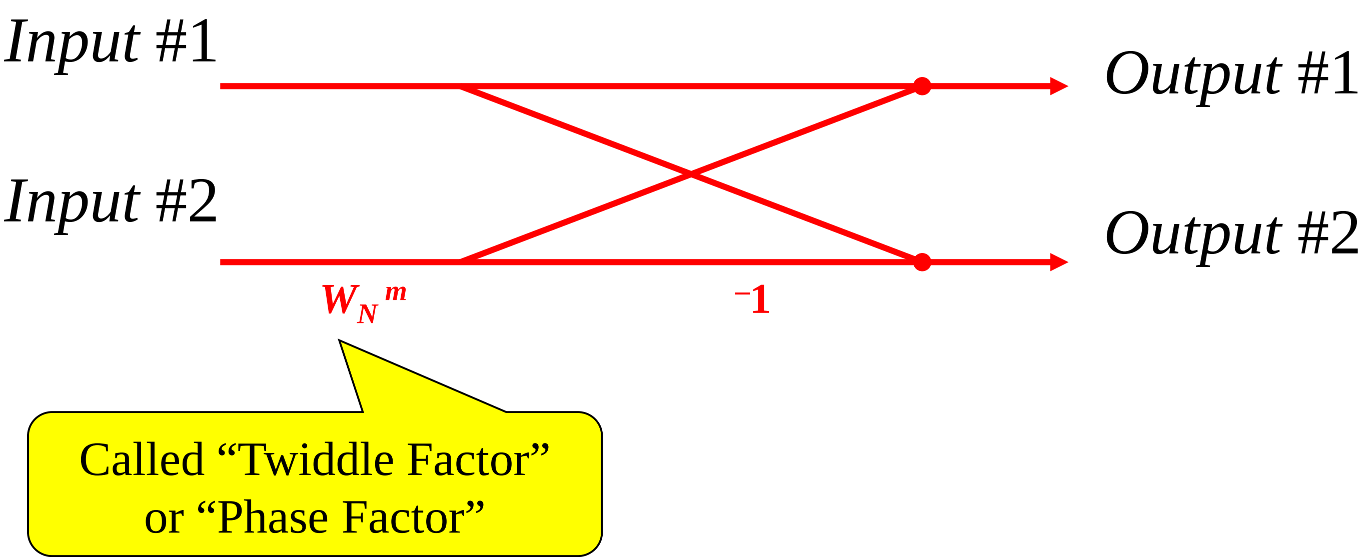
Inherent Structure to Exploit



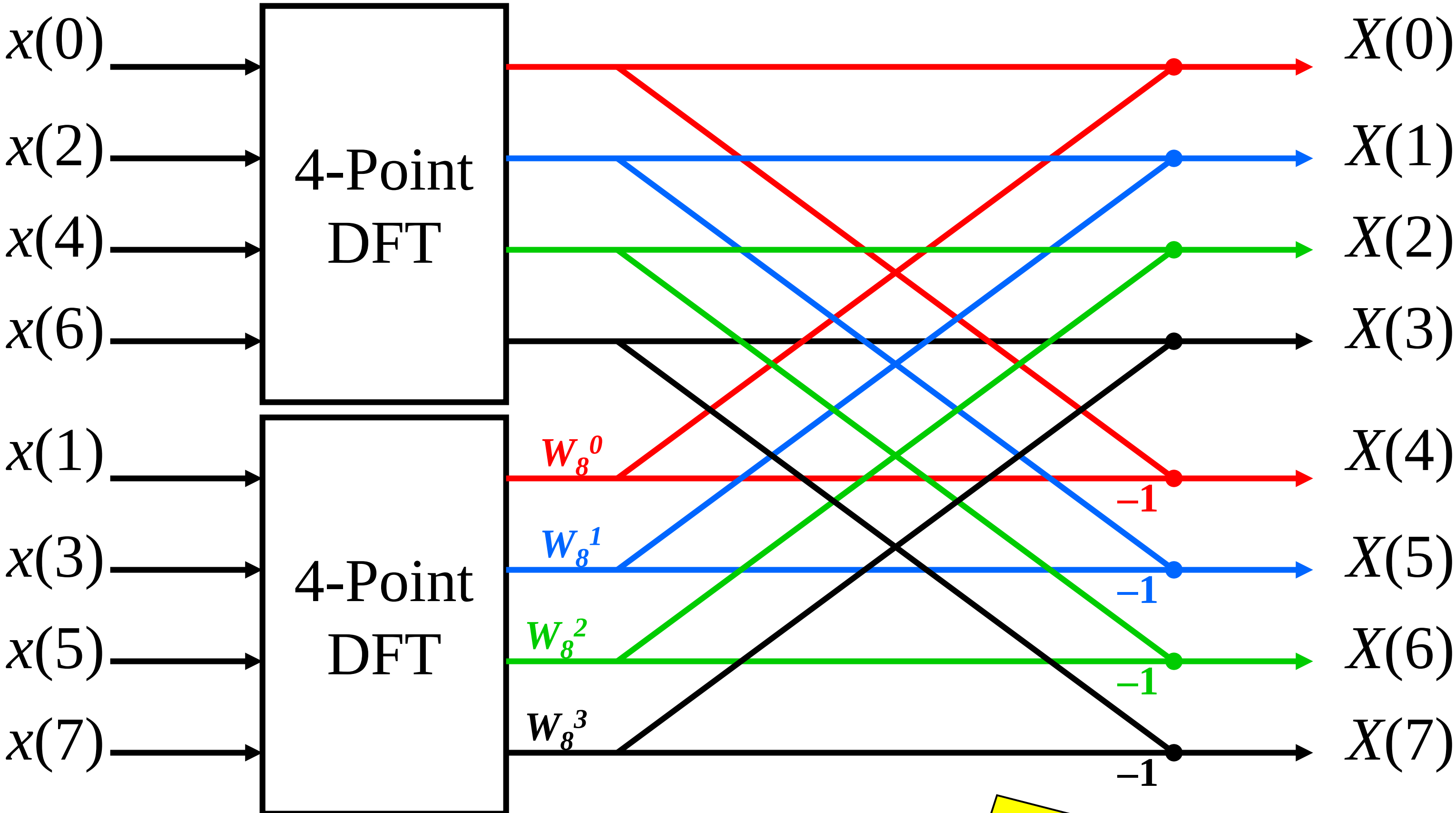
Original Form of "Butterfly": 2 Complex Multiplies



Improved Form of "Butterfly": 1 Complex Multiply



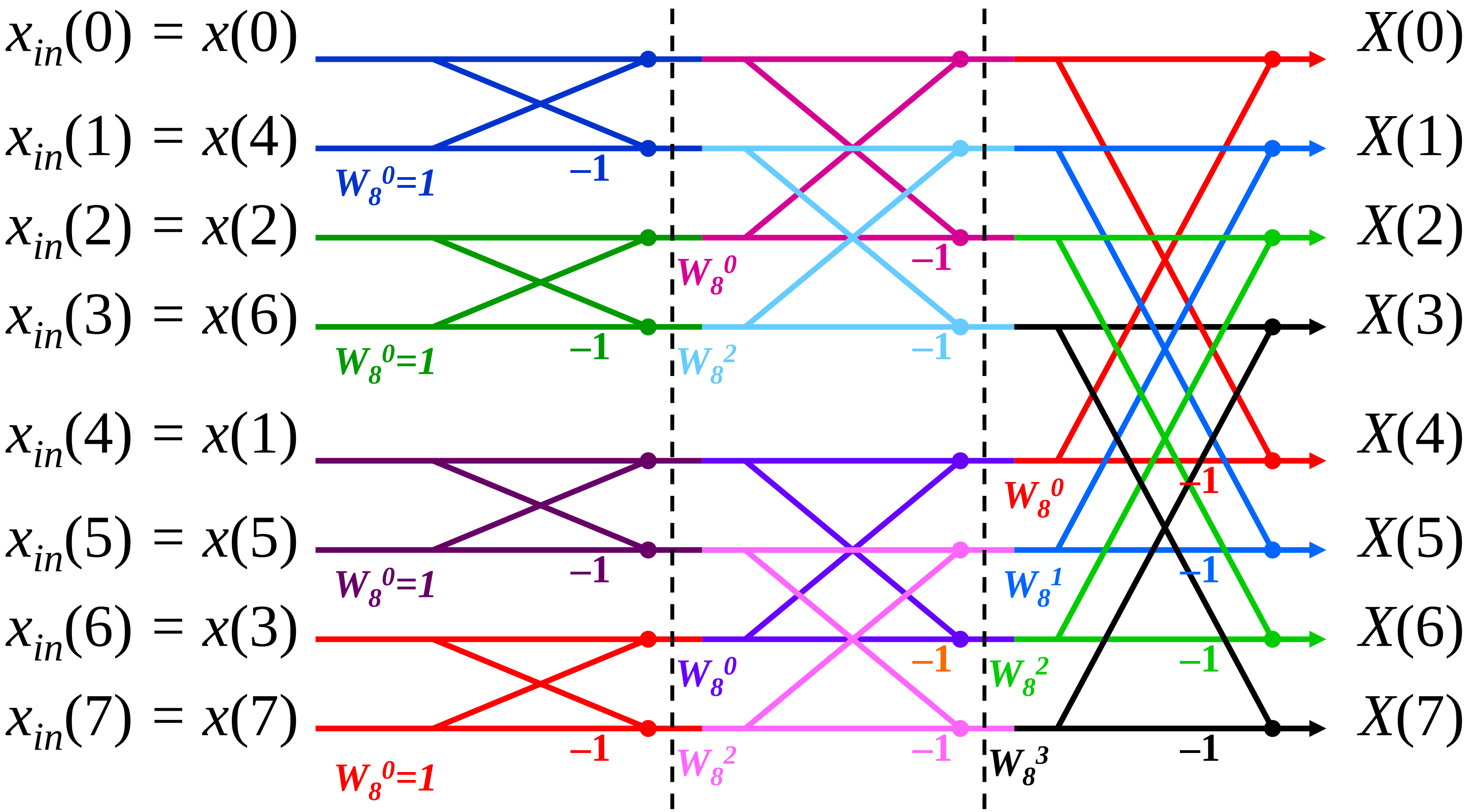
Using Improved Form of "Butterfly"



Now… we can do the same thing to each of these two 4-pt DFTs. Each 4-pt DFT will be decomposed into two 2-pt DFTs that are fused together via Butterflies.
And because $N = 2^v$ we can do this decomposition a total of *v* times!

We see here that this approach involves four 2-pt DFTs (the Butterfly structure)

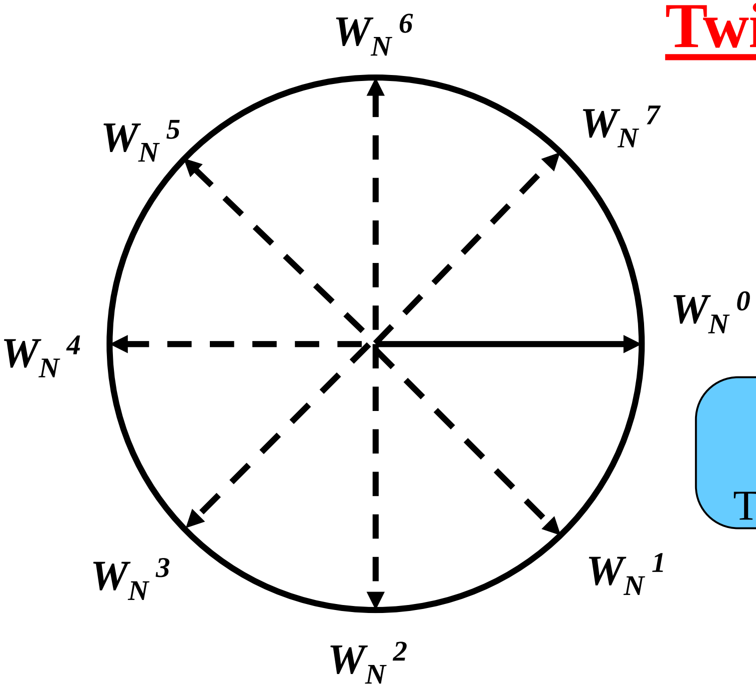
FFT for N=8



1st Stage	2nd Stage	3rd Stage
BF_Step=1	BF_Step=1	BF_Step=1
BF_Span=1	BF_Span=2	BF_Span=4
Blk_Step=2	Blk_Step=4	Blk_Step=8

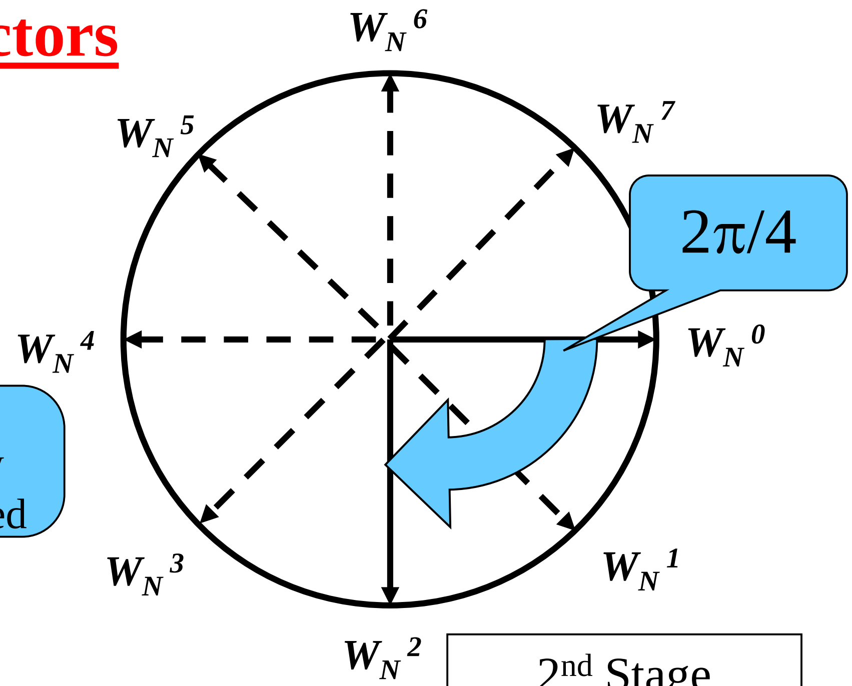
Note that each stage consists of applying Twiddle Factors & then 2-pt DFTs

Determining Twiddle Factors

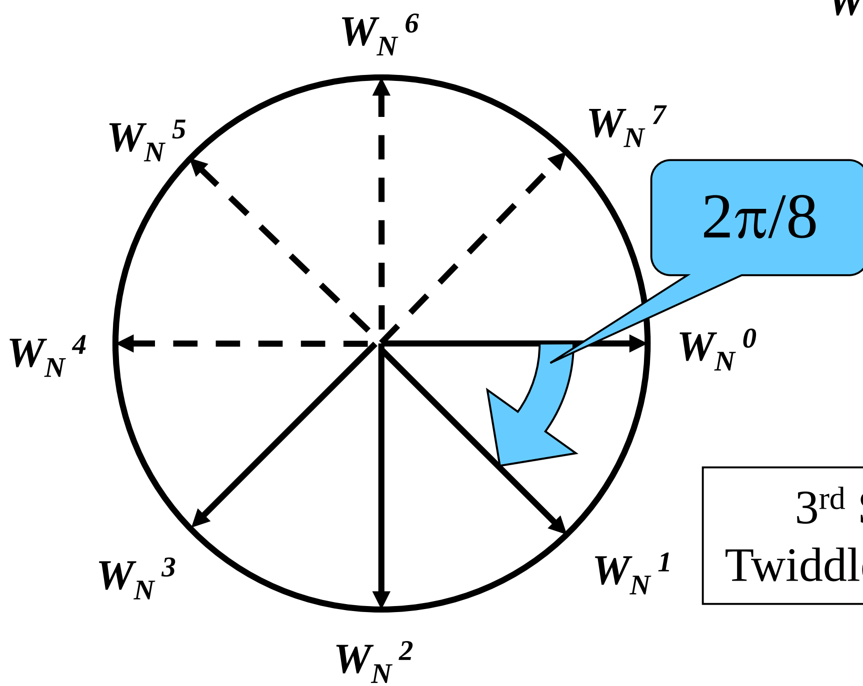


1st Stage Twiddle Factors

Dashed Arrows Show TFs Not Needed



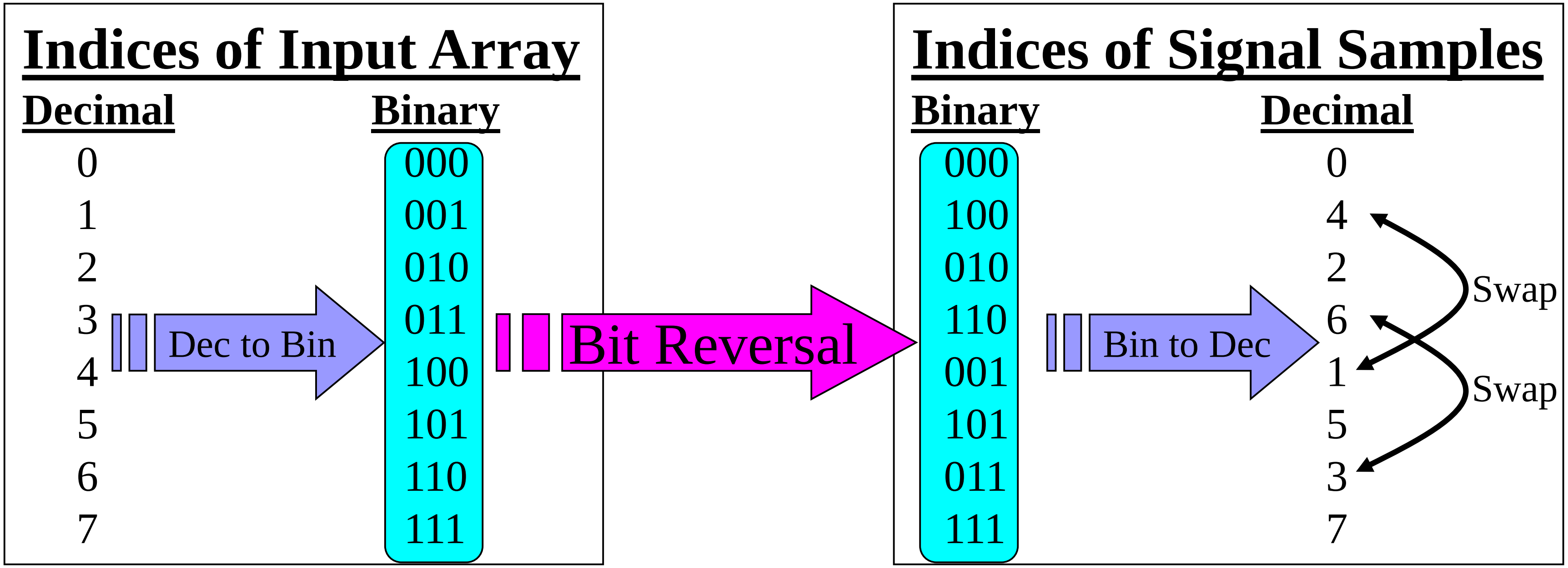
2nd Stage Twiddle Factors



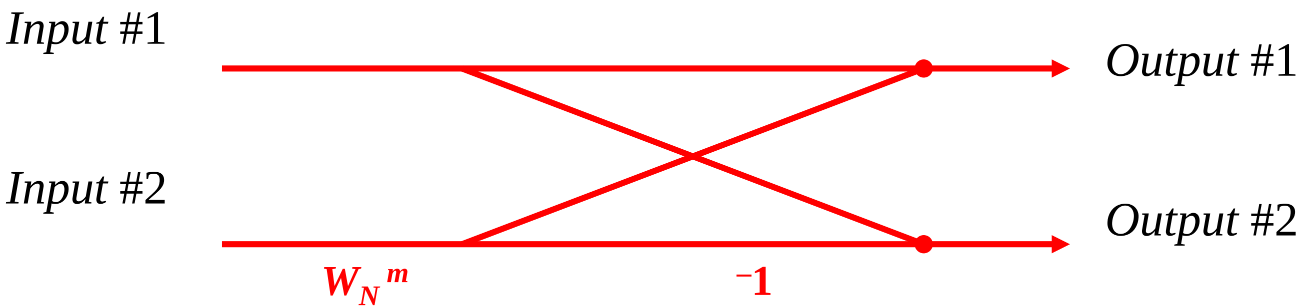
3rd Stage Twiddle Factors

This particular form of the FFT is called ***"Decimate-In-Time"*** because it was developed by dividing the time samples into groups… The result is that the order in which you need the input signal samples is not sequential.

For Decimate-In-Time the inputs are in "Bit Reversed Order":



FFT Computational Complexity



1 Complex Mult/Butterfly

2 Complex Add/Butterfly

N/2 Butterflies/Stage

$\log_2$N Stages/FFT

Complex Mult/FFT = ($\log_2 N$ Stages/FFT)× (N/2 BF/Stage) × (1 Complex Mult/BF)

= $\boldsymbol{N/2 \log_2 N}$ (compared to DFT's N^2)

Complex Adds/FFT = ($\log_2$N Stages/FFT)× (N/2 BF/Stage) × (2 Complex Adds/BF)

= $\boldsymbol{N \log_2 N}$ (compared to DFT's $\approx N^2$)

FFT is **"Order $N \log_2 N$"** or $\boldsymbol{O(N \log_2 N)}$

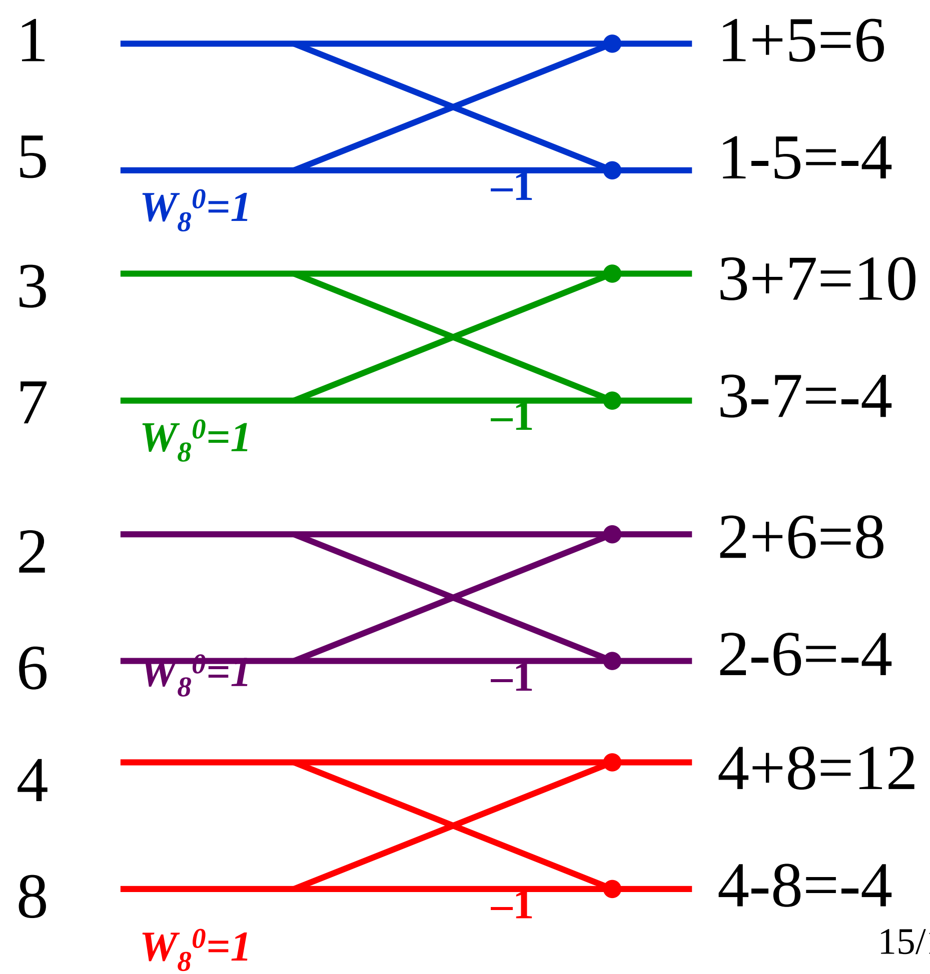
FFT Example

Input

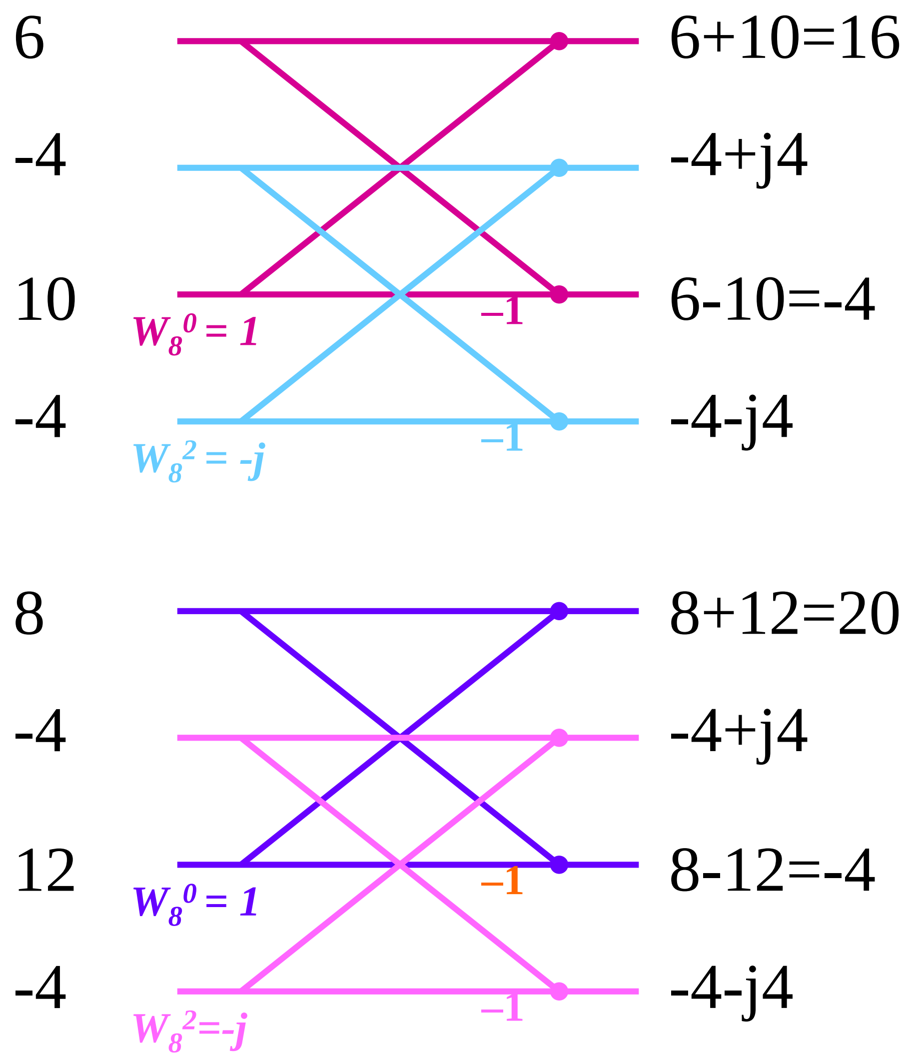
x(n) = [1 2 3 4 5 6 7 8]

n =   0 1 2 3 4 5 6 7

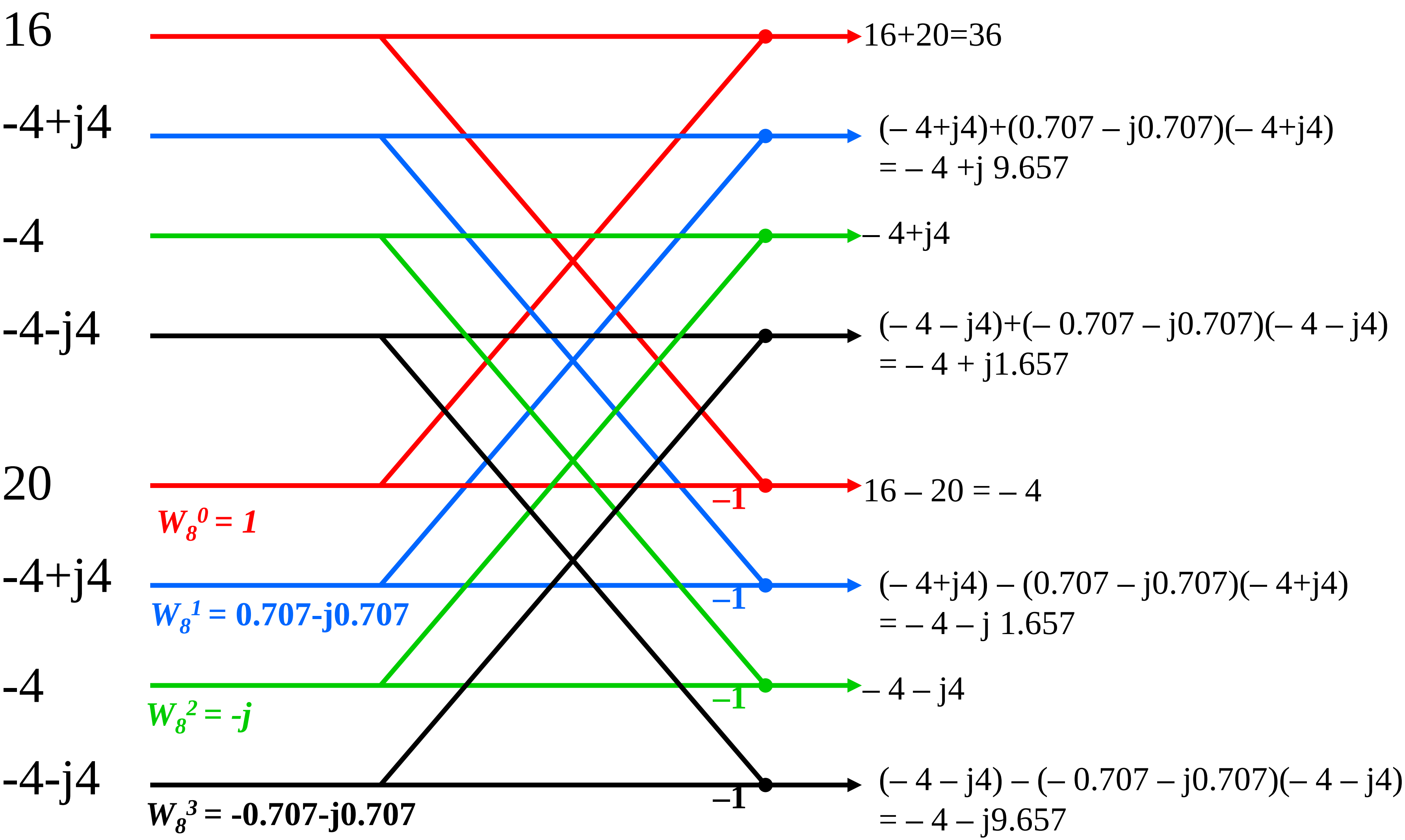
First Stage



Second Stage



Third Stage


16
-4+j4
-4
-4-j4
20
-4+j4
-4
-4-j4
$W_8^0 = 1$
$W_8^1 = 0.707\text{-}j0.707$
$W_8^2 = \text{-}j$
$W_8^3 = \text{-}0.707\text{-}j0.707$
–1
–1
–1
–1
16+20=36
(– 4+j4)+(0.707 – j0.707)(– 4+j4)
= – 4 +j 9.657
– 4+j4
(– 4 – j4)+(– 0.707 – j0.707)(– 4 – j4)
= – 4 + j1.657
16 – 20 = – 4
(– 4+j4) – (0.707 – j0.707)(– 4+j4)
= – 4 – j 1.657
– 4 – j4
(– 4 – j4) – (– 0.707 – j0.707)(– 4 – j4)
= – 4 – j9.657