Remote Sensing Laboratory
Dept. of Information Engineering and Computer Science
University of Trento
Via Sommarive, 14, I-38123 Povo, Trento, Italy

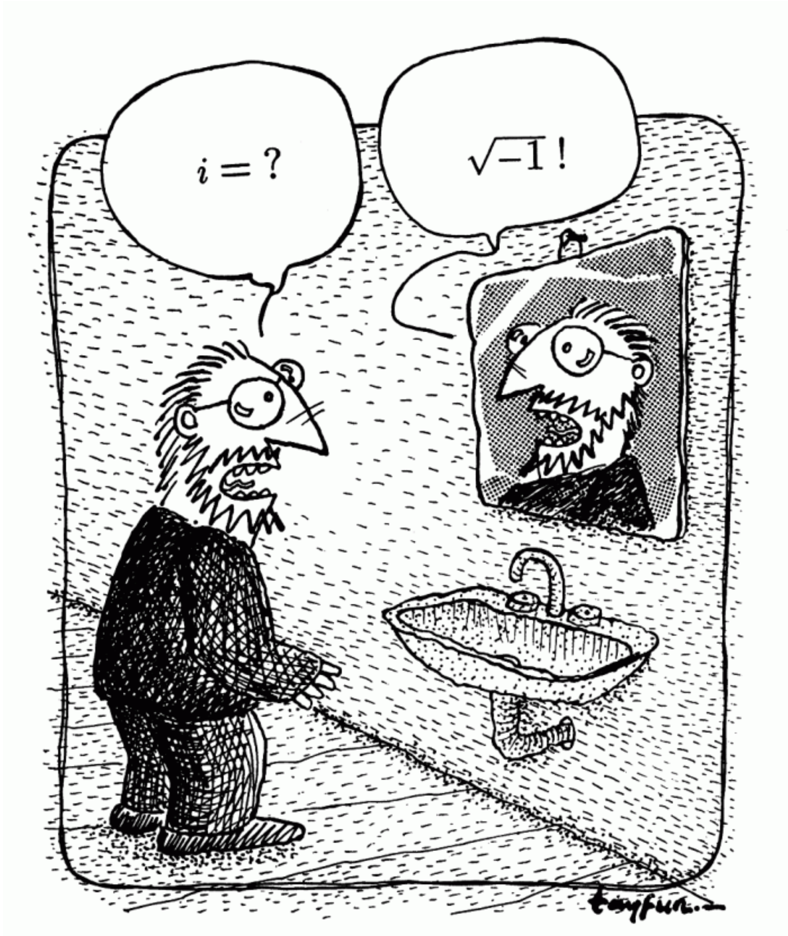
Digital Signal Processing
Lecture 8

Quote of the Day

Any sufficiently advanced technology is indistinguishable from magic.

Arthur C. Clarke

E-mail: demir@disi.unitn.it
Web page: http://rslab.disi.unitn.it


$i = ?$
$\sqrt{-1}\,!$

DFT: Filtering of Long Data Sequences

- The input signal x[n] is often very long especially in real-time signal processing applications.

- There are applications where we need to perform a linear convolution of a finite length sequence with a sequence that is of length that is much greater than that of the first sequence.

- Let h[n] be a finite length sequence of length M and x[n] be a sequence of a finite length much greater than M.

- Our objective is to develop computationally efficient DFT based methods to implement linear convolution of h[n] and x[n].

- The strategy is to segment the input signal into fixed-size blocks prior to processing and to compute DFT based methods for each block separately.

DFT: Filtering of Long Data Sequences

✓ Two approaches to real-time linear filtering of long inputs:

- Overlap-Add Method
- Overlap-Save Method

Overlap Add

- The linear convolution of a discrete-time signal of length L and a discrete-time signal of length M produces a discrete-time convolved result of length L+M-1.

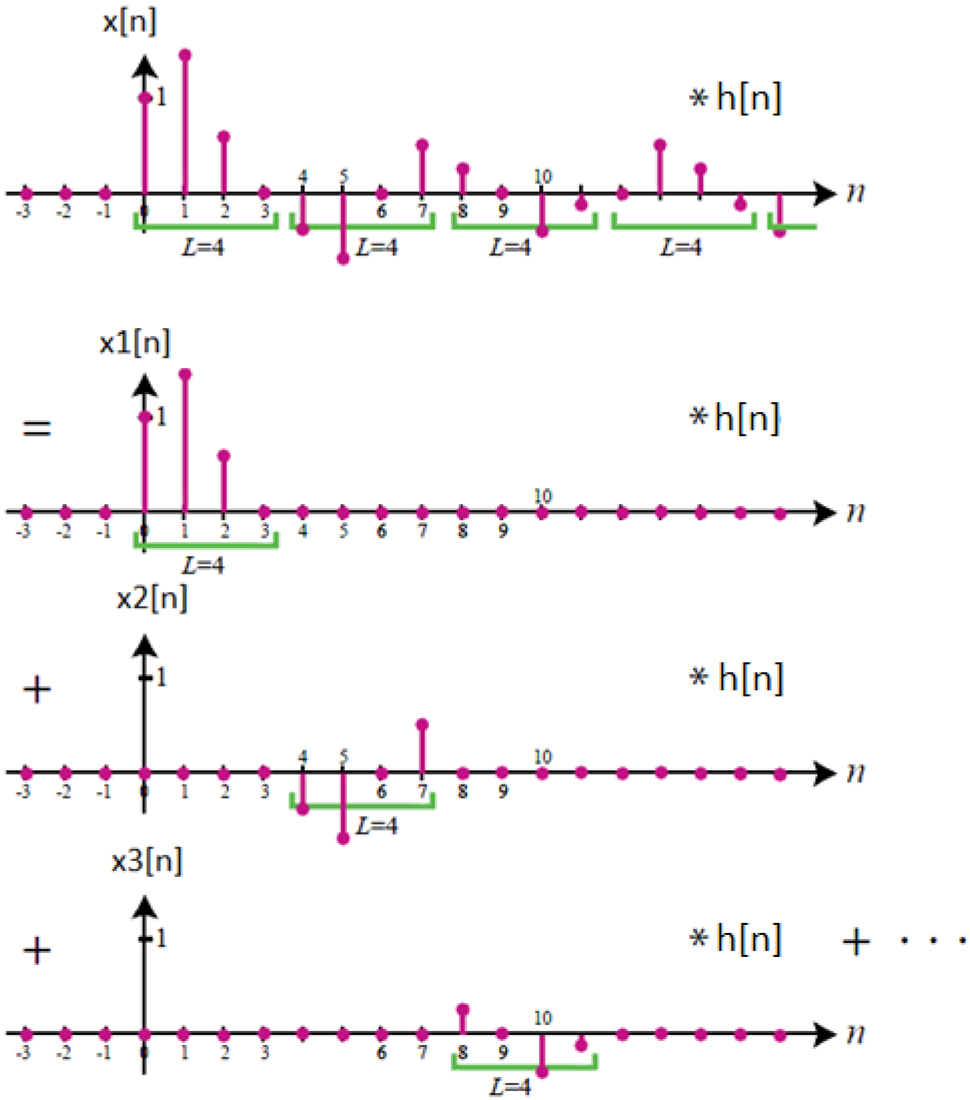
- Remember:

$$\left(x_1[n]+x_2[n]\right)*h[n]=x_1[n]*h[n]+x_2[n]*h[n]$$

- In overlap add method, input signal x[n] is divided into non overlapping blocks $x_m[n]$ with length L.

- Each input block $x_m[n]$ is individually convolved with h[n] to produce the output $y_m[n]$.

Overlap Add

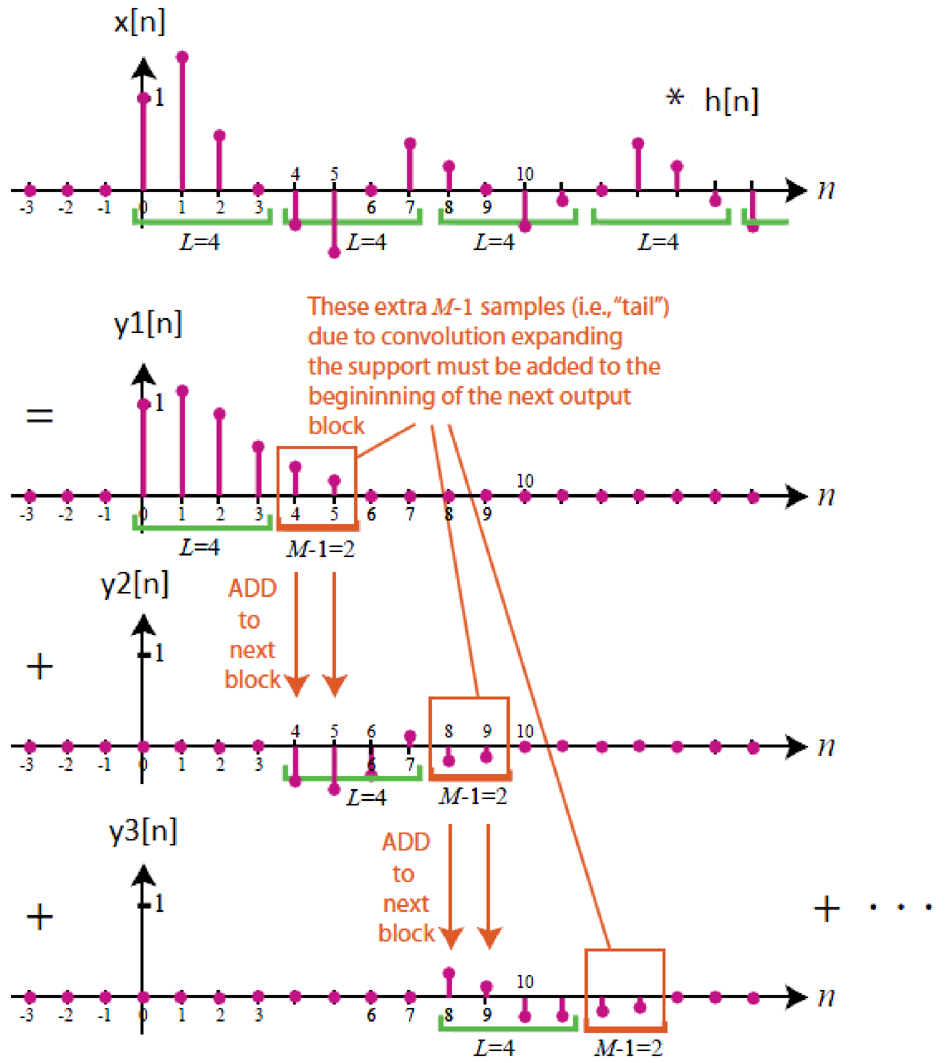


Overlap Add-Filtering

- ✓ Apply N-DFT and N-IDFT where N=L+M-1.
- ✓ Thus, if L and M are smaller than N apply zero padding to $x_m[n]$ and $h[n]$.
- ✓ We set $N = L + M - 1$ (the length of the linear convolution result) and zero pad $x_m[n]$ and $h[n]$ to define $n = 0,1, \dots, N-1$.
- ✓ Using DFT for Linear Convolution:

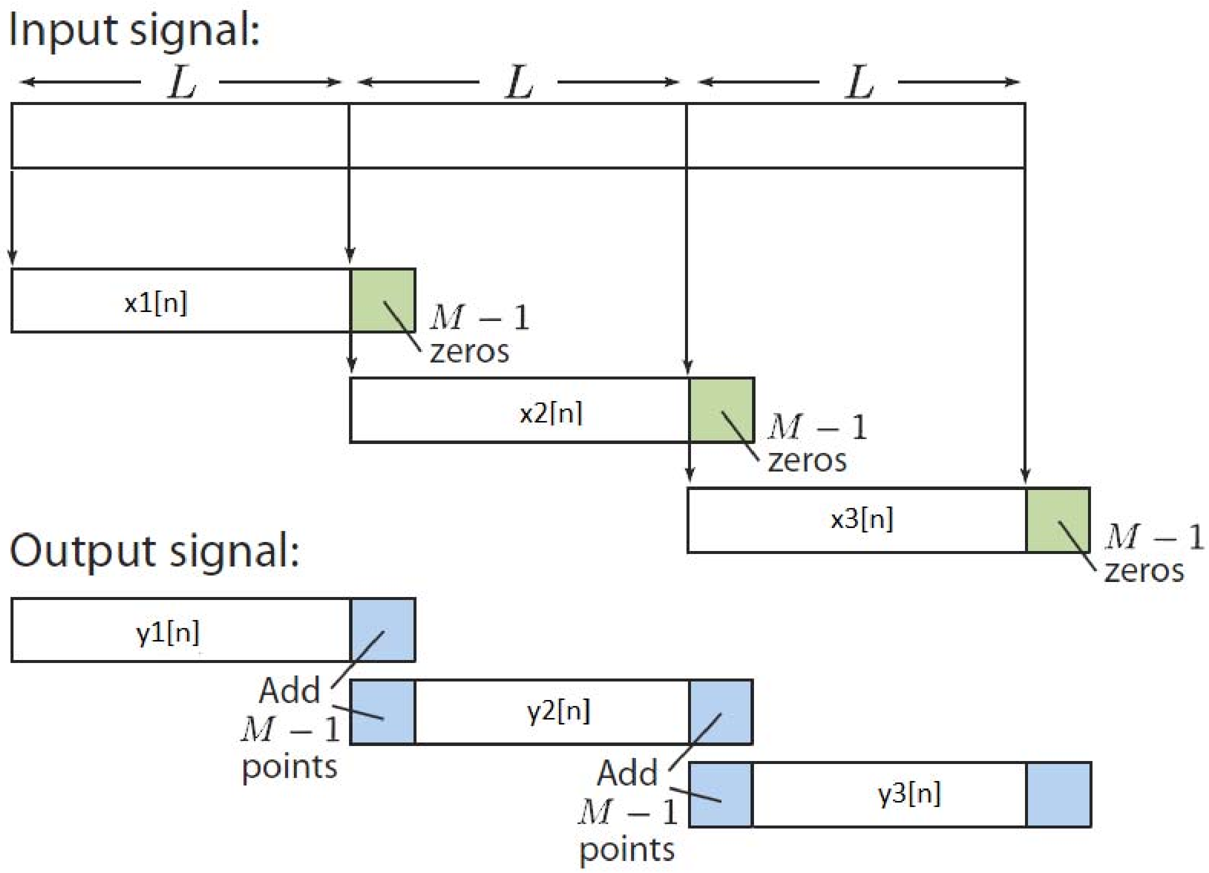
1. Take N-DFT of $x_m[n]$ to give $X_m[k]$, $k = 0,1, \dots, N-1$.
2. Take N-DFT of $h[n]$ to give $H[k]$, $k = 0,1, \dots, N-1$.
3. Multiply: $Y_m[k] = X_m[k]H[k]$, $k = 0,1, \dots, N-1$.
4. Take N-IDFT of $Y_m[k]$ to give $y_m[n]$, $n = 0,1, \dots, N-1$.

Overlap Add



x[n]
* h[n]
L=4
y1[n]
=
These extra M-1 samples (i.e.,"tail") due to convolution expanding the support must be added to the begininning of the next output block
M-1=2
y2[n]
+
ADD to next block
y3[n]
+
+ . . .

Overlap Add



Input signal:
L
L
L
x1[n]
M − 1
zeros
x2[n]
M − 1
zeros
x3[n]
M − 1
zeros
Output signal:
y1[n]
Add
M − 1
points
y2[n]
Add
M − 1
points
y3[n]

Overlap Add

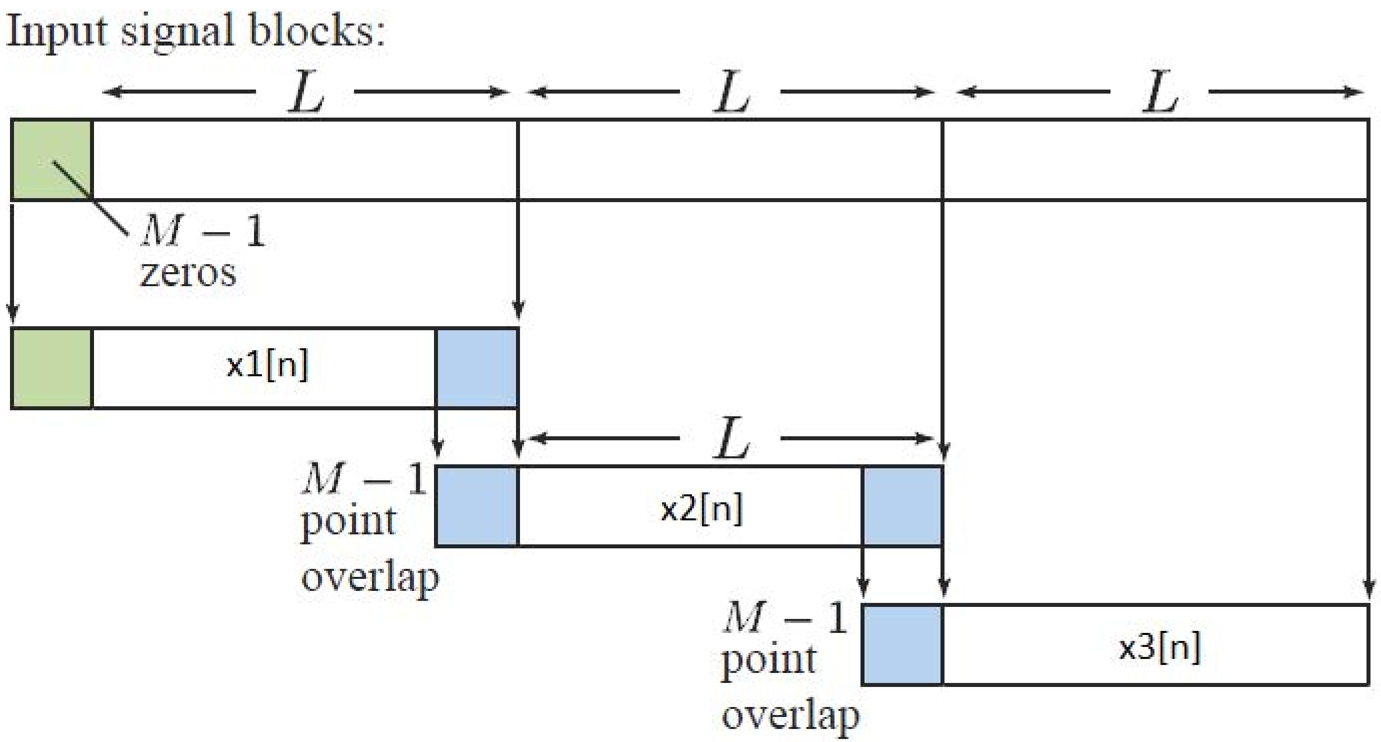
- ✓ Output blocks $y_m[n]$ must be fitted together appropriately to generate: $x[n]*h[n]$

- ✓ If you DO NOT overlap and add, but only append the output blocks $y_m[n]$, then you will not get the true $y[n]$ sequence.

Overlap Save

- ✓ The Overlap-Save method aims at segmenting x[n] into overlapping blocks.
- ✓ Accordingly, all input blocks $x_m[n]$ are of length $N = (L + M - 1)$ and contain sequential samples from x[n].
- ✓ Input block $x_m[n]$ for $m > 1$ overlaps containing the last M-1 points of the previous block $x_{m-1}[n]$.
- ✓ For m = 1, there is no previous block, so the first M-1 points are zeros.

Overlap Save



Overlap Save

$$x_1[n] = \left\{ \underbrace{0, 0, \ldots 0}_{M-1\ zeros}, x[0], x[1], \ldots, x[L-1] \right\}$$

$$x_2[n] = \left\{ \underbrace{x[L-M+1], \ldots, x[L-1]}_{last\ M-1\ \text{samples from } \mathrm{x}_1[\mathrm{n}]}, x[L], \ldots, x[2L-1] \right\}$$

$$x_3[n] = \left\{ \underbrace{x[2L-M+1], \ldots, x[2L-1]}_{last\ M-1\ \text{samples from } \mathrm{x}_2[\mathrm{n}]}, x[2L], \ldots, x[3L-1] \right\}$$

$$\vdots$$

The last M-1 samples from the previous input block must be used for use in the current input block.

Overlap Save

✓ Apply the N-DFT and N-IDFT where: $N = L + M - 1$

- Only one-time zero-padding of $h[n]$ of length $M << L < N$ is required to have $h[n]$ with $n = 0,1,\ldots,N-1$

- The input blocks $x_m[n]$ are of length N to start, so no zero-padding is necessary.

Overlap Save

$N = L + M - 1.$

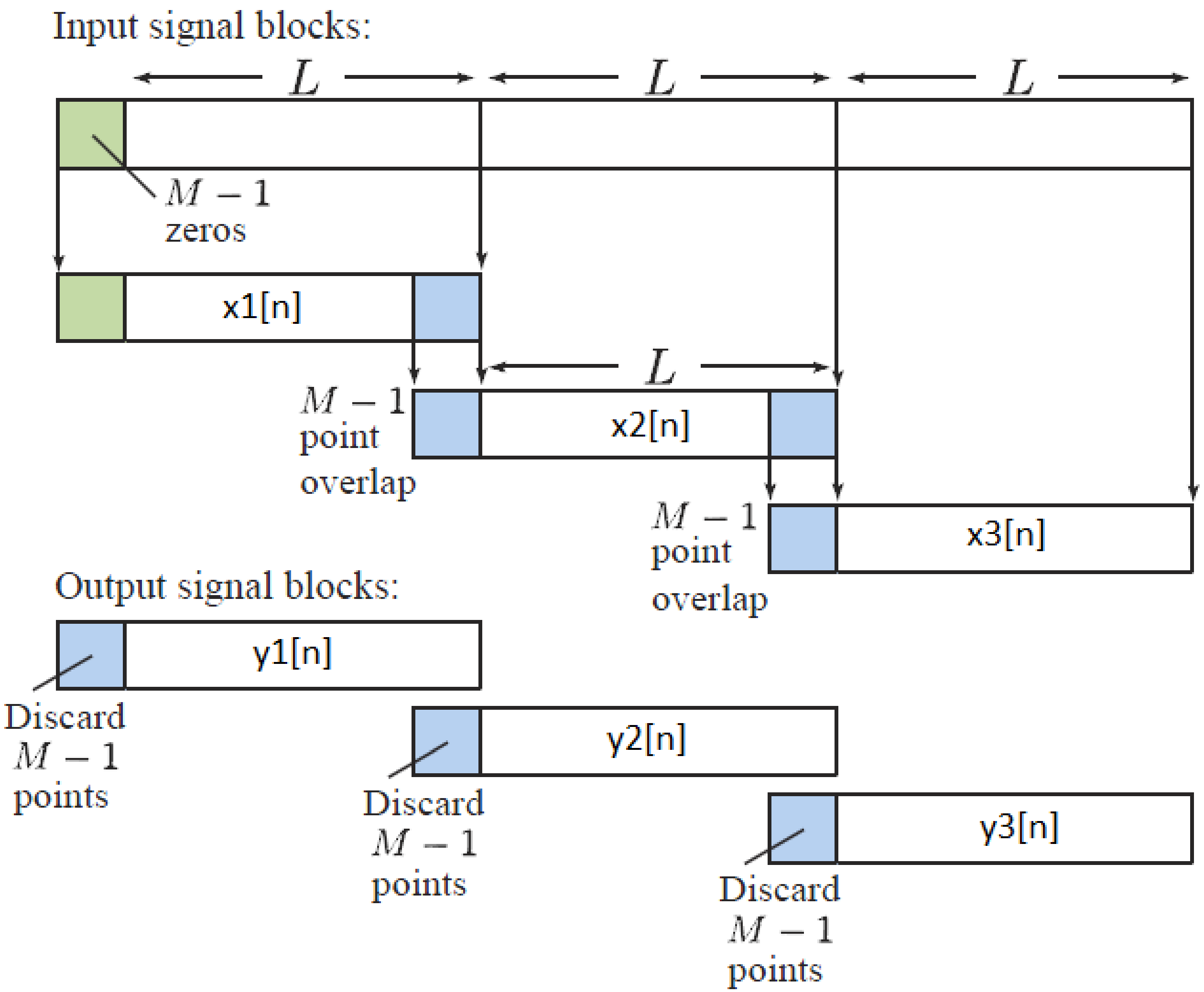
✓ Let $x_m[n]$ is defined for $n = 0,1,\ldots,N-1$ and $h[n]$ is defined for $n = 0,1,\ldots,M-1$

✓ We zero pad $h[n]$ to have $n = 0,1,\ldots,N-1$

1. Take N-DFT of $x_m[n]$ to give $X_m[k]$, $k = 0,1,\ldots,N-1$.
2. Take N-DFT of $h[n]$ to give $H[k]$, $k = 0,1,\ldots,N-1$.
3. Multiply: $Y_m[k] = X_m[k]\,H[k]$, $k = 0,1,\ldots,N-1$.
4. Take N-IDFT of $Y_m[k]$ to give $y_m[n]$, $n = 0,1,\ldots,N-1$.

✓ The first M -1 points of each output block are discarded due to overcome with aliasing. The remaining L points of each output block are appended to form y[n].

Overlap Save



Discrete Fourier Transform (DFT)

- ✓ Despite the fact that the DFT is a vector with finite size, computing it for large values of N is very intensive. In order to compute each

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j\frac{2\pi}{N}kn}, \quad k = 0,1,\ldots N-1$$

N complex multiplications and N-1 complex additions (that is computationally expensive) for each k are needed.

- ✓ Thus DFT has $O(N^2)$ computations, whereas it is possible to reduce it to $O(N\log_2 N)$ by fast Fourier Transform (FFT) algorithm .

Fast Fourier Transform(FFT) Algorithm

✓ It is numerically efficient way to calculate DFT;

✓ It is not another Fourier Representation, but it is an algorithm.

N	1000	10^6	10^9
N^2	10^6	10^{12}	10^{18}
$N\log_2 N$	10^4	$20\text{x}10^6$	$30\text{x}10^9$

Lets suppose that each operation takes 1 ns

$30\text{x}10^9$ns ~30 seconds

10^{18}ns ~31.2 years

Fast Fourier Transform (FFT) Algorithm

- FFT exploits
  - symmetry of the complex exponential:

$$W_N^{k+\frac{N}{2}} = -W_N^k \longrightarrow W_N^{k+\frac{N}{2}} = e^{-j2\pi\frac{k+N/2}{N}} = e^{-j2\pi\frac{k}{N}} e^{-j2\pi\frac{N/2}{N}} = e^{-j2\pi\frac{k}{N}} e^{-j\pi}$$

$$= e^{-j2\pi\frac{k}{N}} (\cos(-\pi) + j\sin(-\pi)) = e^{-j2\pi\frac{k}{N}} (-1) = -W_N^k$$

  - periodicity of the complex exponential:

$$W_N^{k+N} = W_N^k \longrightarrow W_N^{k+N} = e^{-j2\pi\frac{k+N}{N}} = e^{-j2\pi\frac{k}{N}} e^{-j2\pi\frac{N}{N}} = e^{-j2\pi\frac{k}{N}} e^{-j2\pi}$$

$$= e^{-j2\pi\frac{k}{N}} (\cos(-2\pi) + j\sin(-2\pi)) = e^{-j2\pi\frac{k}{N}} = W_N^k$$

- The FFT algorithm relies on the fact that the task of computing the N-point DFT of a signal can be broken down into two tasks, each involving an N/2-point DFT.

Radix-2 FFT

- We will demonstrate how to exploit the symmetry and periodicity of W_N^k:
  - to make an N-Point DFT look like two $N/2$-Point DFTs;
  - to make an $N/2$-Point DFT look like two $N/4$-Point DFTs;
  - to make an $N/4$-Point DFT look like two $N/8$-Point DFTs;
- The halving of the DFT length each time gives the name Radix-2 FFT.

Note: We use the convention N-DFT to specify an N-Point DFT.

Radix-2 FFT

Two strategies:

- Decimation in time (our focus in the lecture)
- Decimation in frequency

- Note: We assume that N is a power of two; i.e., $N = 2^r$.

FFT Algorithm -Decimation in Time

- ✓ It aims to build a big DFT from smaller ones.
- ✓ Assume $N=2^m$
- ✓ Separate x[n] into two sequence of length N/2
  - – Even indexed samples in the first sequence
  - – Odd indexed samples in the other sequence

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn}$$

$$= \sum_{\text{n even}}^{N-1} x[n] e^{-j(2\pi/N)kn} + \sum_{\text{n odd}}^{N-1} x[n] e^{-j(2\pi/N)kn}$$

FFT Algorithm -Decimation in Time

even indices

odd indices

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

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

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

N/2 point DFT of x[2r]

N/2 point DFT of x[2r]

G[k] and H[k] are the N/2-point DFT's of each subsequence

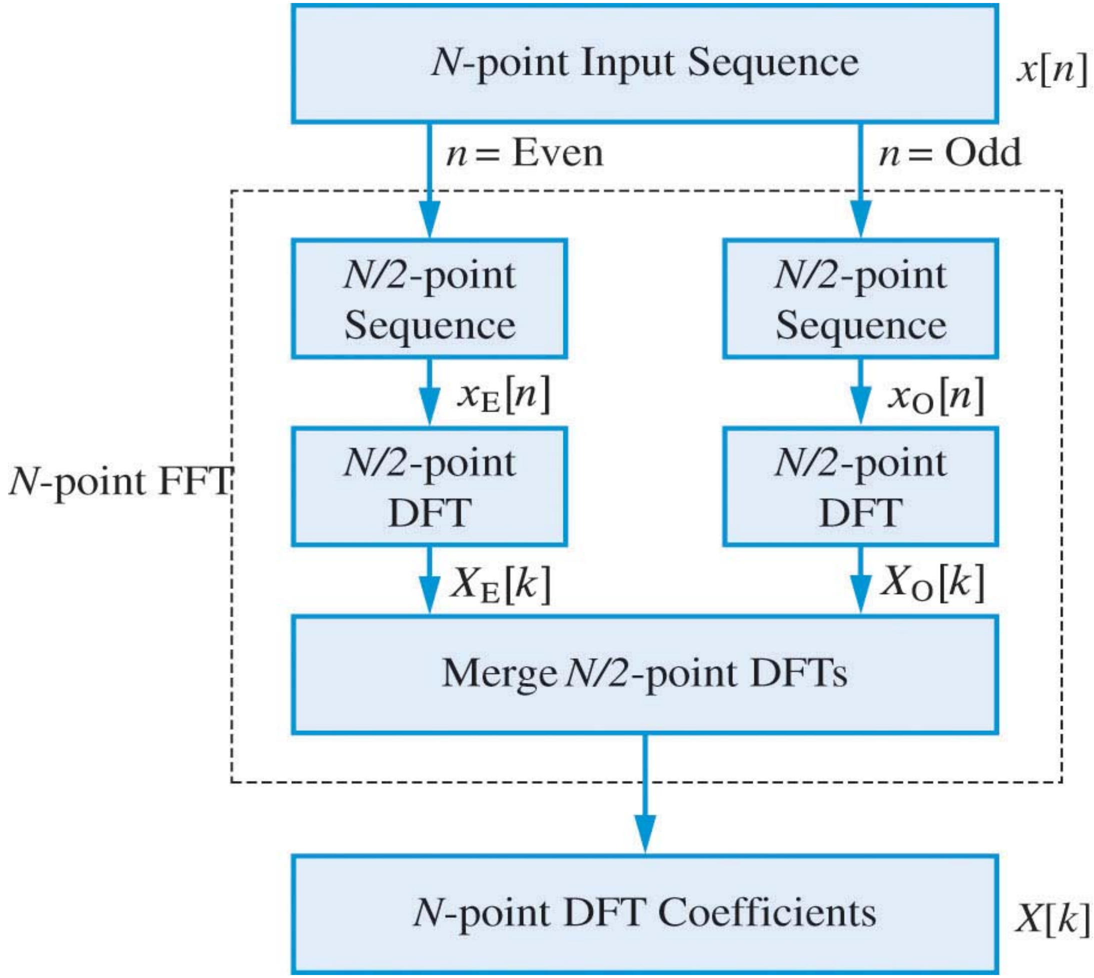
FFT Algorithm -Decimation in Time

$$X[k] = G[k] + W_N^k H[k], \; 0 \le k \le \left(N/2-1\right)$$

$$X[k+\frac{N}{2}] = G[k+\frac{N}{2}] + W_N^{k+\frac{N}{2}} H[k+\frac{N}{2}] = G[k] - W_N^k H[k], \; 0 \le k \le \left(N/2-1\right)$$

$$W_N^{k+\frac{N}{2}} = e^{-j\frac{2\pi}{N}\left(\mathrm{k}+\frac{N}{2}\right)} = e^{-j\frac{2\pi}{N}(\mathrm{k})} e^{-j\frac{2\pi}{N}\left(\frac{N}{2}\right)} = e^{-j\frac{2\pi}{N}(\mathrm{k})}(-1) = -W_N^k$$

FFT Algorithm -Decimation in Time



FFT Algorithm -Decimation in Time

DFT Computation

$$a[n] = x[2n] \xrightarrow[N/2]{\text{DFT}} A[k] = \sum_{n=0}^{(N/2)-1} a[n] W_{N/2}^{kn}$$

$$0 \leq k \leq \frac{N}{2} - 1$$

$$b[n] = x[2n+1] \xrightarrow[N/2]{\text{DFT}} B[k] = \sum_{n=0}^{(N/2)-1} b[n] W_{N/2}^{kn}$$

DFT Merging

$$X[k] = A[k] + W_N^k B[k]$$

$$X\left[k + \frac{N}{2}\right] = A[k] - W_N^k B[k]$$

Holds for every even N !

Complexity:

$$2\left(\frac{N}{2}\right)^2 \simeq \frac{N^2}{2} \Rightarrow$$

50% reduction

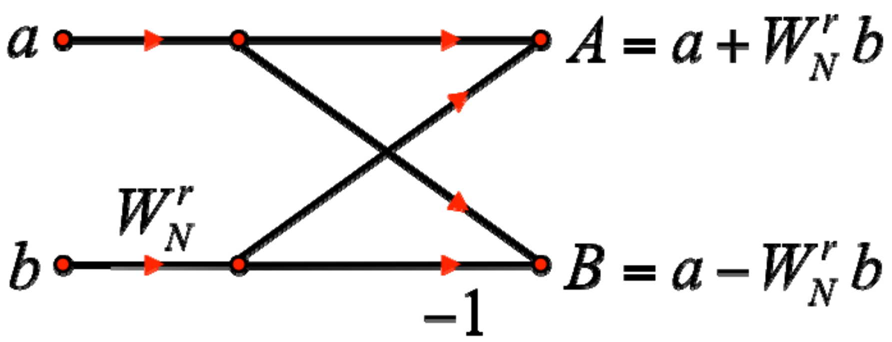
FFT Algorithm -Decimation in Time

$\{x[0]\ x[1]\ x[2]\ x[3]\ x[4]\ x[5]\ x[6]\ x[7]\}$

$\{a[0]\ a[1]\ a[2]\ a[3]\} = \{x[0]\ x[2]\ x[4]\ x[6]\}$

$\{b[0]\ b[1]\ b[2]\ b[3]\} = \{x[1]\ x[3]\ x[5]\ x[7]\}$

$$A[k] = \sum_{n=0}^{3} a[n] W_4^{kn} \qquad B[k] = \sum_{n=0}^{3} b[n] W_4^{kn}$$

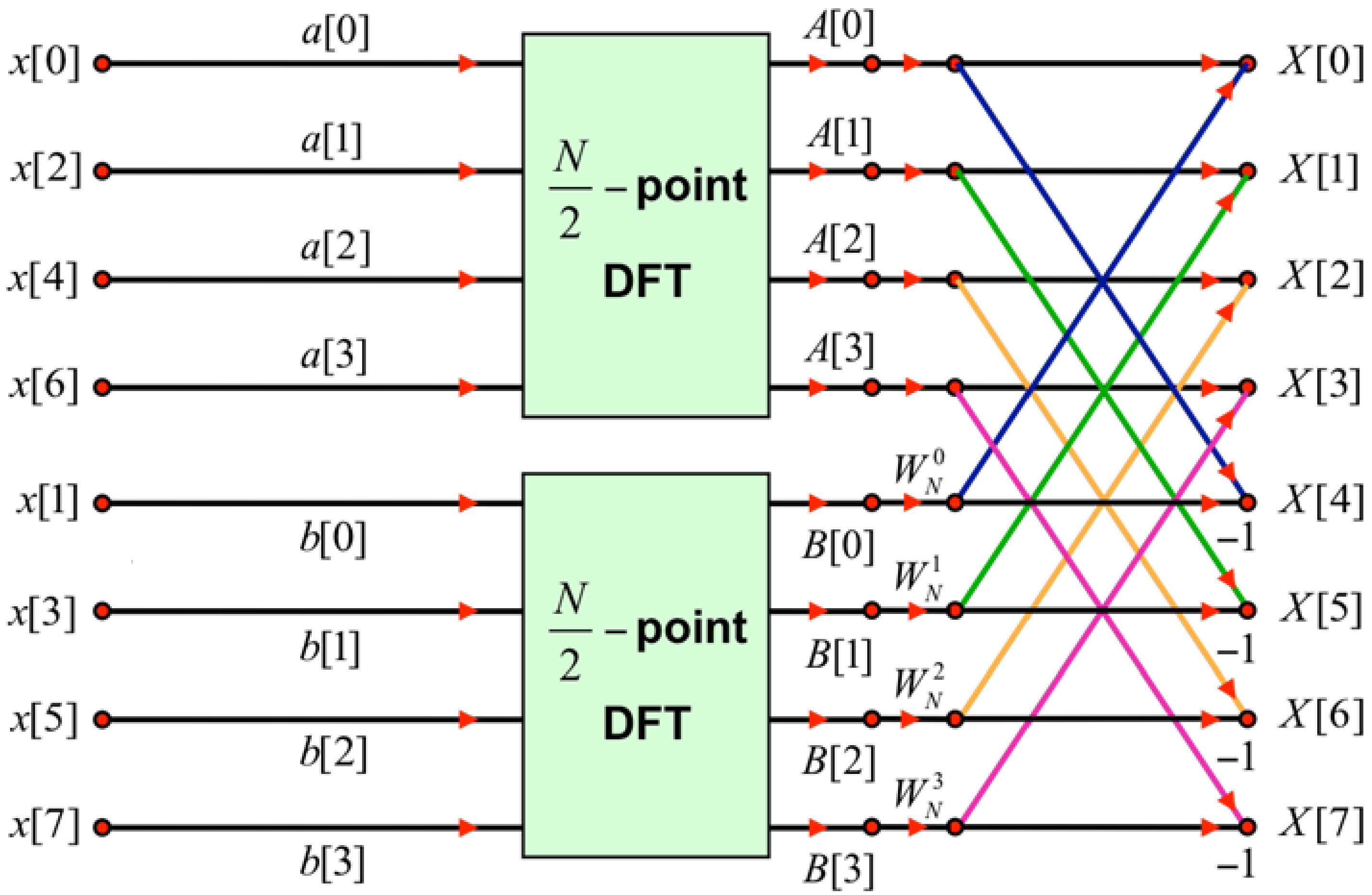


Basic computation flow graph:
DIT butterfly

$$X[k] = A[k] + W_N^k B[k]$$
$$X\left[k + N/2\right] = A[k] - W_N^k B[k]$$

$$X[0] = A[0] + W_8^0 B[0]$$
$$X[4] = A[0] - W_8^0 B[0]$$
$$X[1] = A[1] + W_8^1 B[1]$$
$$X[5] = A[1] - W_8^1 B[1]$$
$$X[2] = A[2] + W_8^2 B[2]$$
$$X[6] = A[2] - W_8^2 B[2]$$
$$X[3] = A[3] + W_8^3 B[3]$$
$$X[7] = A[3] - W_8^3 B[3]$$

FFT Algorithm -Decimation in Time



✓If we keep splitting, we can reduce the computational complexity more.

FFT Algorithm -Decimation in Time

$$\{a[0]\ a[1]\ a[2]\ a[3]\} = \{x[0]\ x[2]\ x[4]\ x[6]\}$$

$$\{c[0]\ c[1]\} = \{a[0]\ a[2]\} = \{x[0]\ x[4]\}$$

$$\{d[0]\ d[1]\} = \{a[1]\ a[3]\} = \{x[2]\ x[6]\}$$

$$A[k] = C[k] + W_4^k D[k]$$
$$A[k+2] = C[k] - W_4^k D[k]$$
$$k = 0,1$$

$$C[k] = c[0]W_2^0 + c[1]W_2^k,\ k = 0,1$$

$$C[0] = c[0] + c[1] = x[0] + x[4]$$
$$C[1] = c[0] - c[1] = x[0] - x[4]$$

$$D[0] = d[0] + d[1] = x[2] + x[6]$$
$$D[1] = d[0] - d[1] = x[2] - x[6]$$

$$A[0] = C[0] + W_4^0 D[0]$$
$$A[2] = C[0] - W_4^0 D[0]$$
$$A[1] = C[1] + W_4^1 D[1]$$
$$A[3] = C[1] - W_4^1 D[1]$$

$$W_{N/2}^k = W_N^{2k} \Rightarrow W_4^0 = W_8^0,\ W_4^1 = W_8^2$$

FFT Algorithm -Decimation in Time

$$\{b[0]\ b[1]\ b[2]\ b[3]\} = \{x[1]\ x[3]\ x[5]\ x[7]\}$$

$$\{e[0]\ e[1]\} = \{b[0]\ b[2]\} = \{x[1]\ x[5]\}$$

$$\{f[0]\ f[1]\} = \{b[1]\ b[3]\} = \{x[3]\ x[7]\}$$

$$B[k] = E[k] + W_4^k F[k]$$
$$B[k+2] = E[k] - W_4^k F[k]$$
$$k = 0,1$$

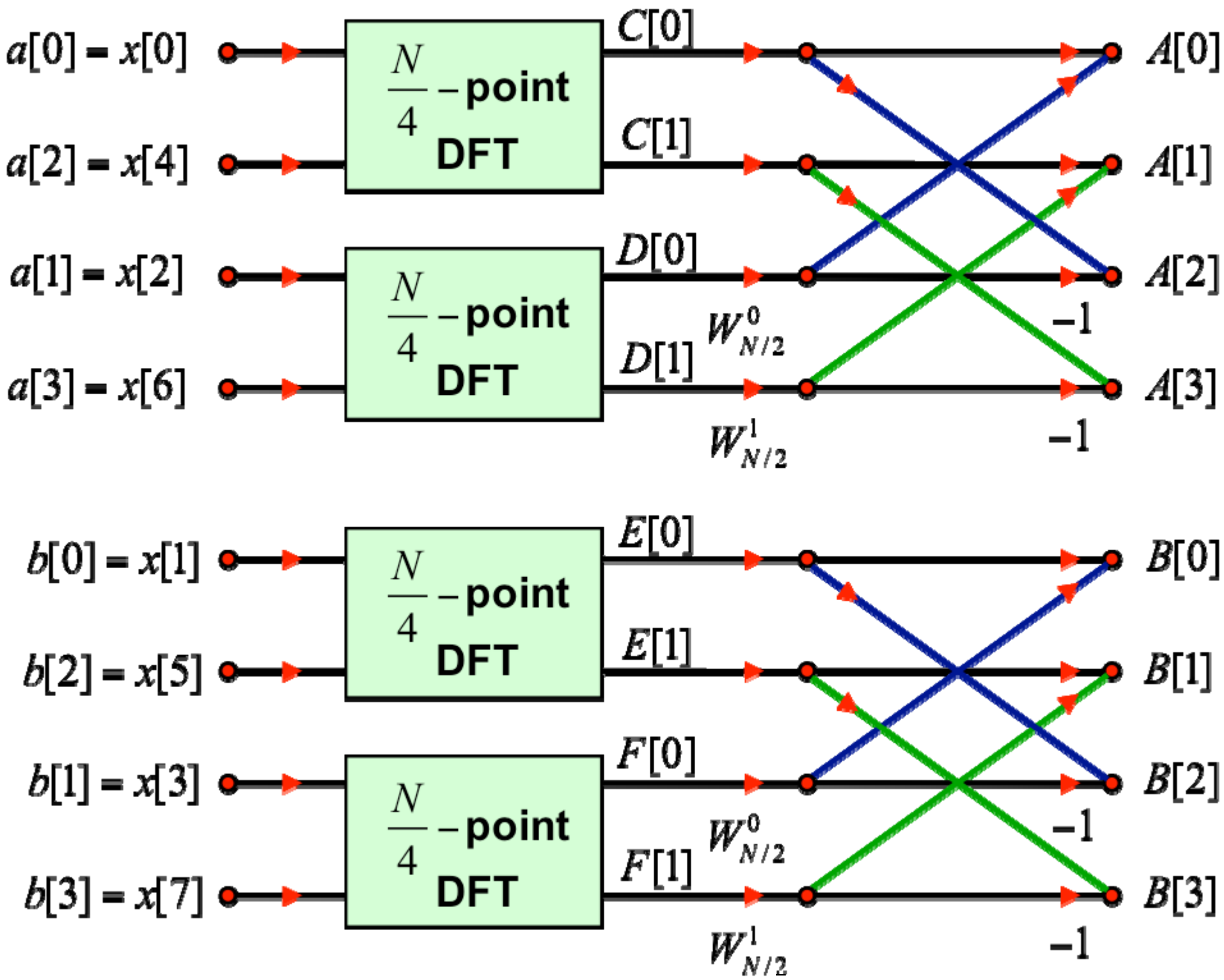
$$E[0] = e[0] + e[1] = x[1] + x[5]$$
$$E[1] = e[0] - e[1] = x[1] - x[5]$$

$$B[0] = E[0] + W_4^0 F[0]$$
$$B[2] = E[0] - W_4^0 F[0]$$
$$B[1] = E[1] + W_4^1 F[1]$$
$$B[3] = E[1] - W_4^1 F[1]$$

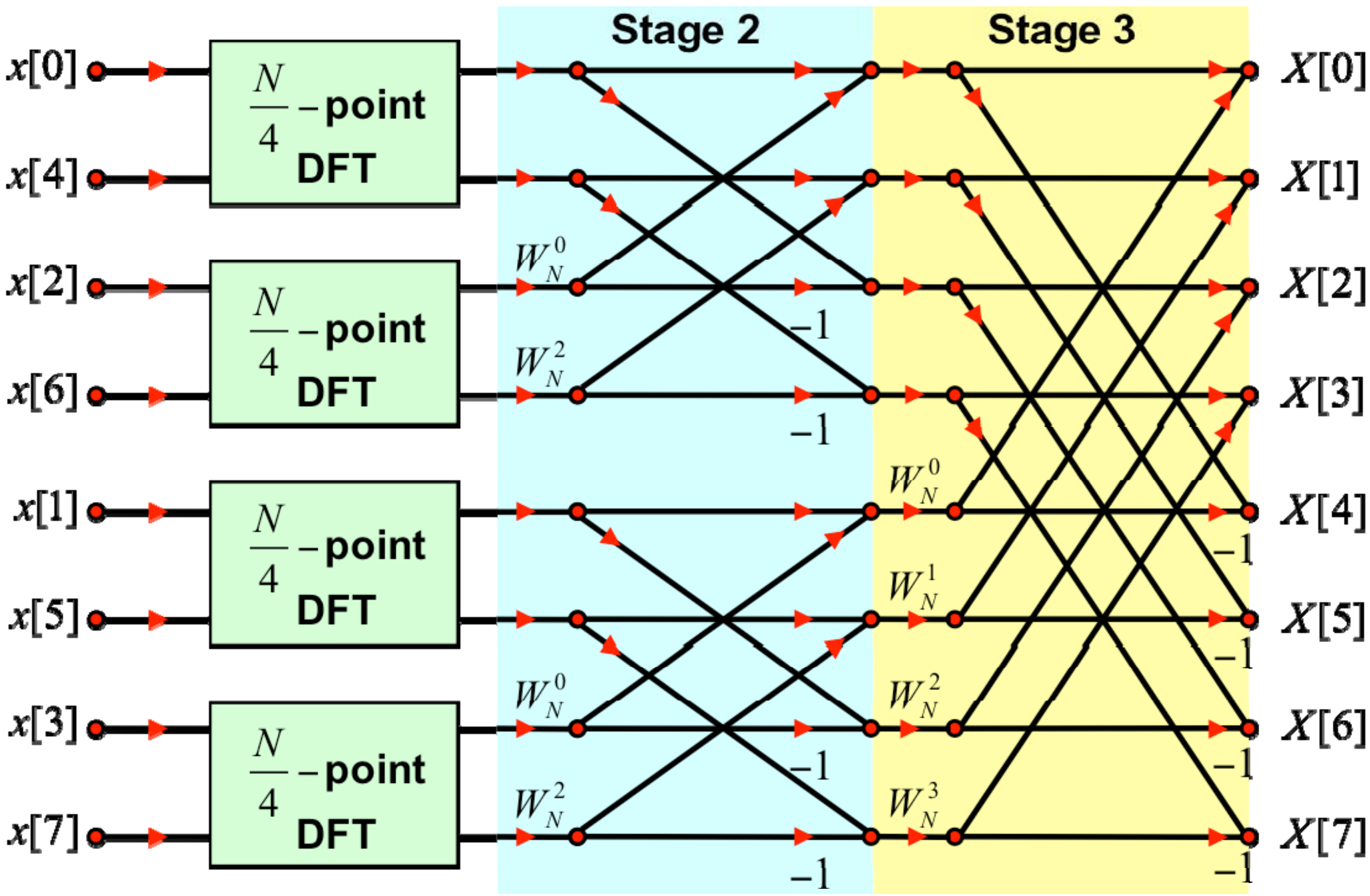
$$F[0] = f[0] + f[1] = x[3] + x[7]$$
$$F[1] = f[0] - f[1] = x[3] - x[7]$$

$$W_{N/2}^k = W_N^{2k} \Rightarrow W_4^0 = W_8^0,\ W_4^1 = W_8^2$$

FFT Algorithm -Decimation in Time



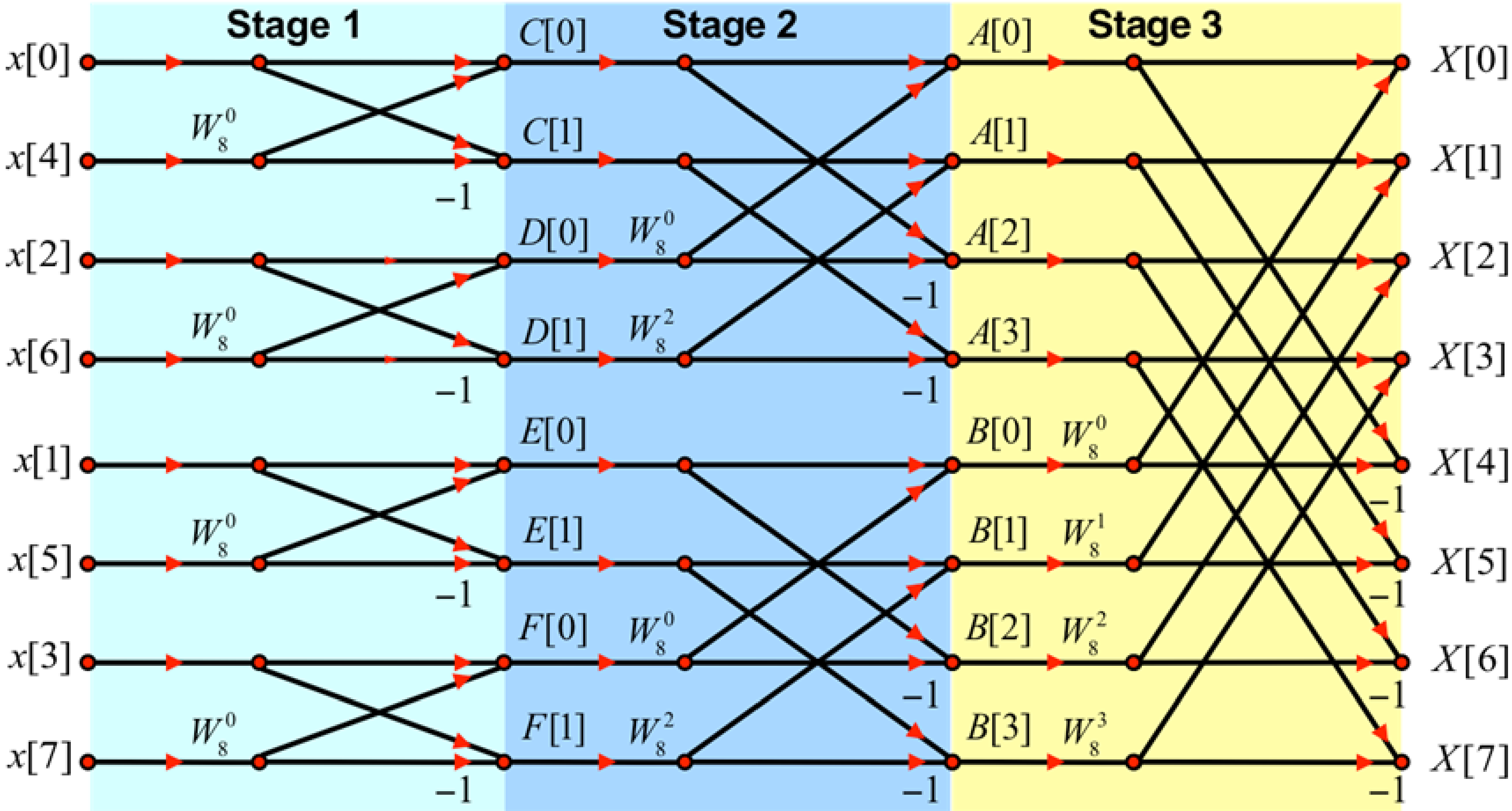
FFT Algorithm -Decimation in Time



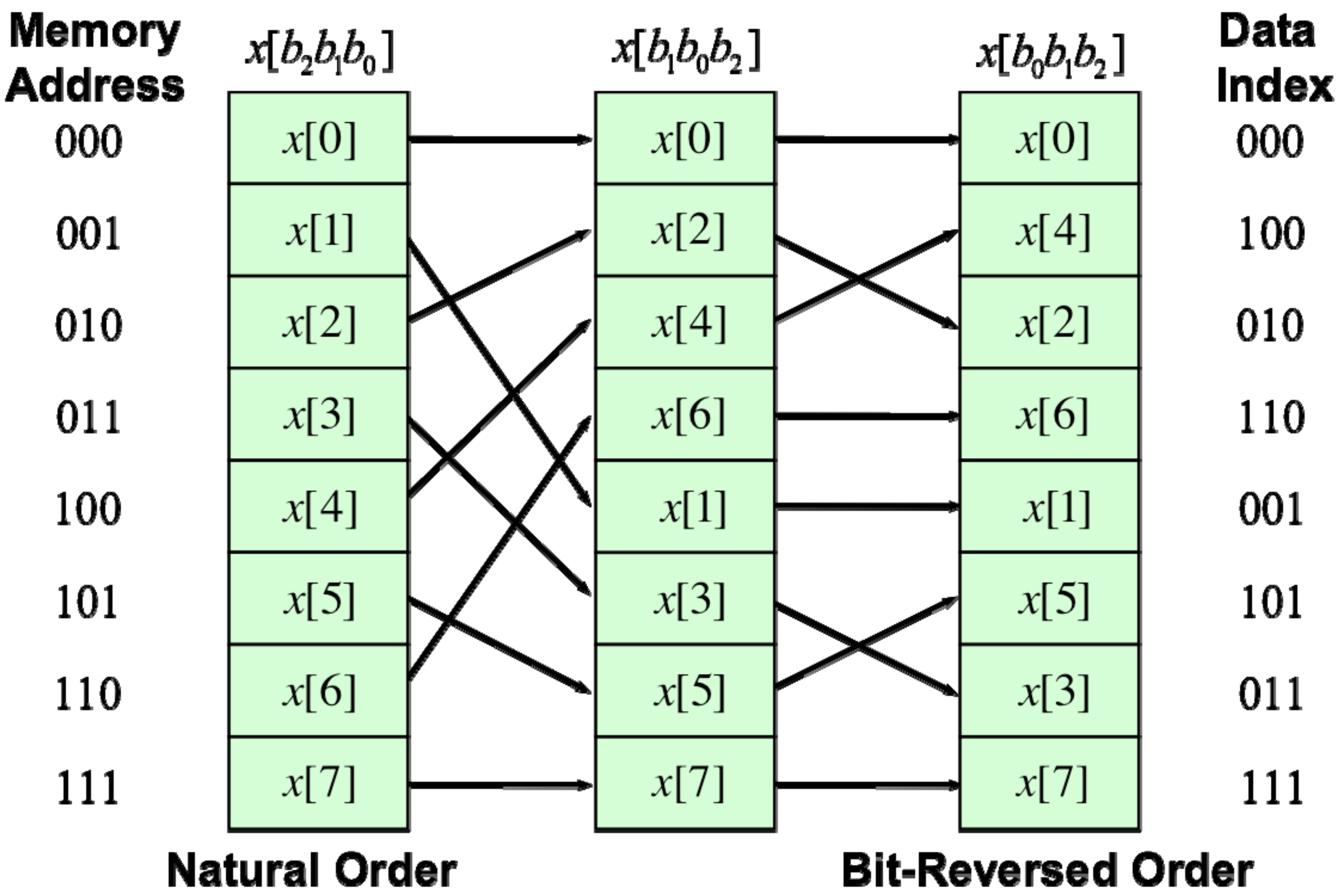
1-point DFT: $X[0] = x[0]$ **2-point DFT:** $X[0] = x[0] + x[1]$

$X[1] = x[0] - x[1]$

FFT Algorithm -Decimation in Time



FFT Algorithm -Decimation in Time

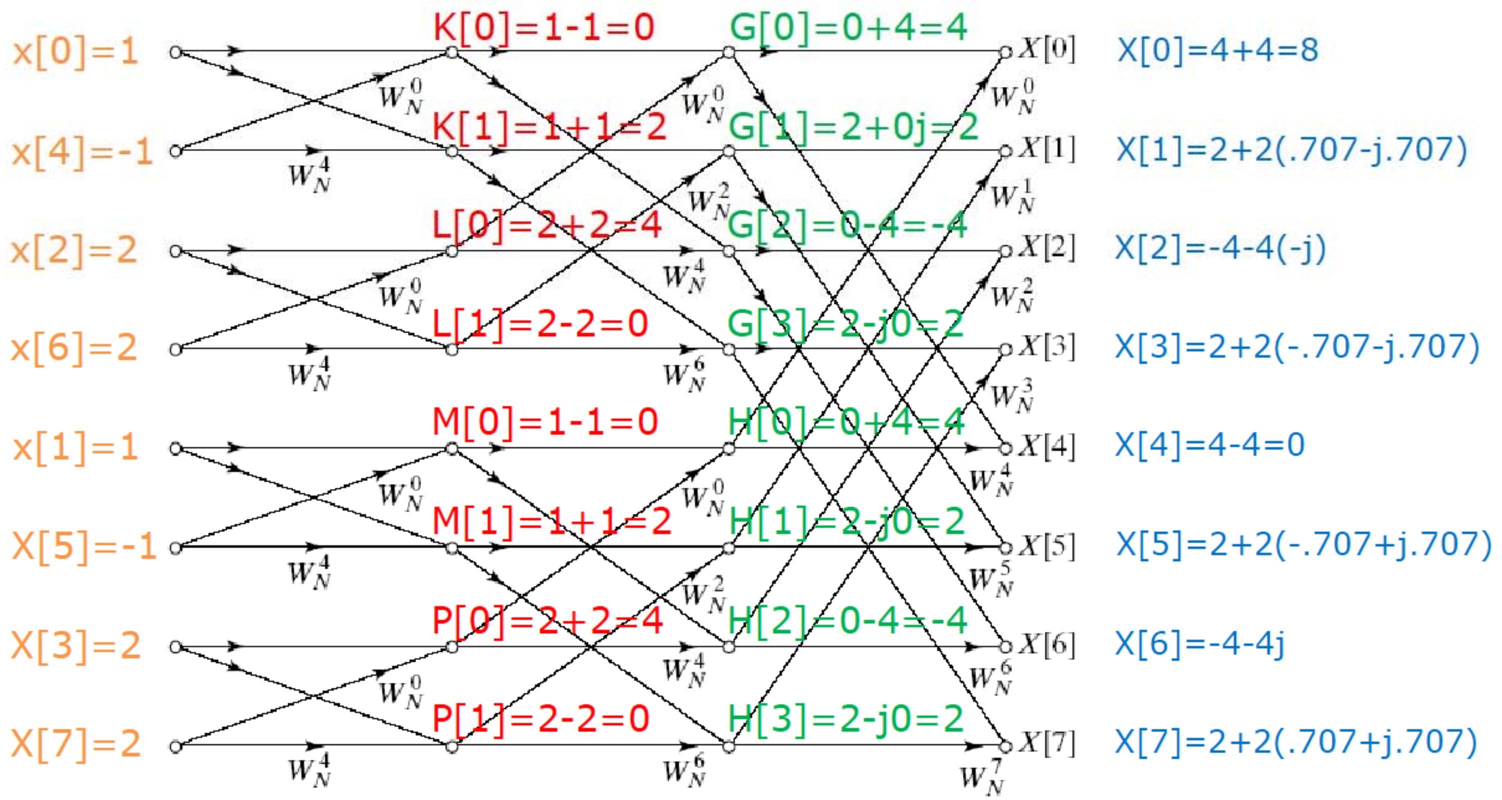


$$n = b_2 2^2 + b_1 2^1 + b_0 2^0$$

$$x[b_2b_1b_0] \rightarrow b_0b_1b_2$$

FFT Algorithm-Decimation in Time

Example: Find the DFT coefficients of x[n] = [1 1 2 2 -1 -1 2 2] using N=8 point FFT.



Fast Fourier Transform Algorithm

- ✓ FFT algorithm that we have included is called radix-2 FFT, as the butterfly sell has 2 inputs and 2 outputs;
- ✓ More efficient FFT can be implemented if using other radix numbers;
- ✓ For Radix-4 FFT, the basic operation unit is based on 4 variables (4 inputs and 4 outputs)

Summary

- Direct computation of the N-point DFT using the defining formula

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

  requires $\mathrm{O}(N^2)$ complex operations.
- Fast Fourier Transform (FFT) algorithms reduce the computational complexity from $\mathrm{O}(N^2)$ to $\mathrm{O}(N \log_2 N)$ operations.
- The decimation-in-time radix-2 FFT algorithm, which requires $N = 2^\nu$, is widely used because it is easy to derive, simple to program, and extremely fast.
- For many years the time for the computation of FFT algorithms was dominated by multiplications and additions. The performance of FFTs on current computers depends upon the structure of the algorithm, the compiler, and the architecture of the machine.

Advice for FFT Users

- FFT is a fast algorithm for the computation of DFT
- We strongly recommend that you only use FFTs with length N a power of 2. If N is not a power of 2, use zero-padding
- For computationally demanding applications use professionally developed code