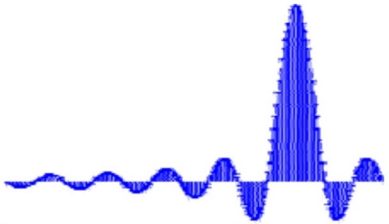
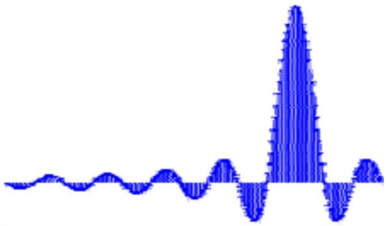


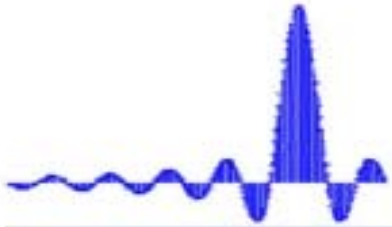
TODAY IN DSP

- Discrete Fourier Transform
  - Analysis
  - Synthesis
  - Orthogonality of discrete complex exponentials
- Periodicity of DFT
- Circular Shift
- Properties of DFT
- Circular Convolution
  - Linear vs. circular convolution
- Computing convolution using DFT
- Some examples
- The DFT matrix
- The relationship between DTFT vs DFT: Reprise



DFT

- DTFT is an important tool in digital signal processing, as it provides the spectral content of a discrete time signal.
  - However, the computed spectrum, $X(\omega)$ is a continuous function of ω, and therefore cannot computed using a computer (the freqz function computes only an approximation of the DTFT, not the actual DTFT).
  - We need a method to compute the spectral content of a discrete time signal and have a spectrum – actually a discrete function – so that it can be computed using a digital computer
- A straightforward solution: Simply sample the frequency variable ω of the DTFT in frequency domain in the $[0\ 2\pi]$ interval.
  - If we want, say N points in the frequency domain, then we divide ω in the $[0\ 2\pi]$ interval into N equal intervals.
  - Then the discrete values of ω are $0, 2\pi/N, 2.2\pi/N, 3.2\pi/N, \ldots, (N-1).\ 2\pi/N$



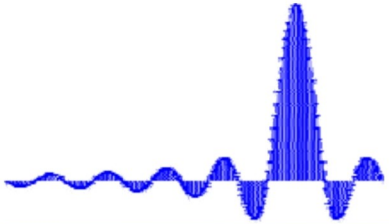
Orthogonal Transforms

$$X[k] = \sum_{n=0}^{N-1} x[n]\psi^*[k,n], \qquad 0 \le k \le N-1$$

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[n]\psi[k,n], \qquad 0 \le n \le N-1$$

with basis sequences: $\psi[k,n]$

which are orthogonal if $\frac{1}{N}\sum_{n=0}^{N-1}\psi[k,n]\psi^*[l,n]\begin{cases}1 & l=k\\ 0 & l\neq k\end{cases}$



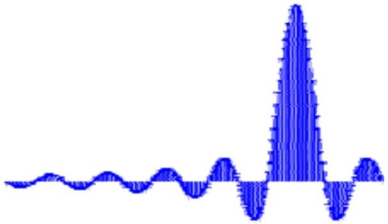
DFT Analysis

- **Definition** - The simplest relation between a length-N sequence x[n], defined for $0 \le n \le N-1$, and its DTFT $X(\omega)$ is obtained by uniformly sampling $X(\omega)$ on the ω-axis $0 \le \omega \le 2\pi$ at $\omega_k = 2\pi k/N$, $0 \le k \le N-1$

- From the definition of the DTFT we thus have

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-2\pi knj/N} = \sum_{n=0}^{N-1} x[n] e^{-\omega_k nj}$$

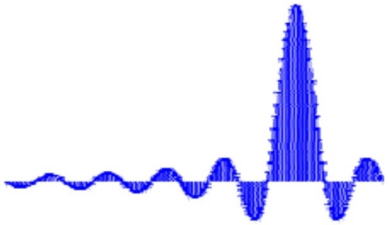
DFT analysis equation



DFT Analysis

- Note the following:
  - ***k*** replaces ω as the frequency variable in the discrete frequency domain
  - ***X*[*k*]** is also (usually) a length-N sequence in the frequency domain, just like the signal x[n] is a length-N sequence in the time domain.
  - X[k] can be made to be longer than N points (as we will later see)
  - The sequence ***X*[*k*]** is called the ***discrete Fourier transform (DFT)*** of the sequence ***x*[*n*]**
- Using the notation $W_N = e^{-j2\pi/N}$ the DFT is usually expressed as:

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



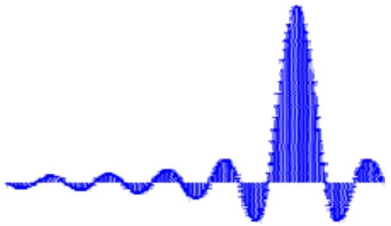
Inverse DFT (DFT Synthesis)

- The inverse discrete Fourier transform, also known as the synthesis equation, is given as

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

DFT synthesis equation

- To verify the above expression we multiply both sides of the above equation by $W_N^{\ell n} = e^{j\frac{2\pi}{N}n\ell}$ and sum the result from $n = 0$ to $n = N-1$



Orthogonality of Discrete Exponentials

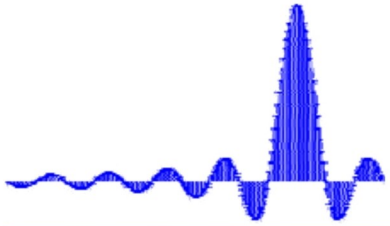
- During the proof, multiplication by $W_N^{\ell n} = e^{j\frac{2\pi}{N}n\ell}$ will result in an expression that includes summation of several discrete exponentials, which can be computed using the result of the following lemma
- Lemma: **Orthogonality of discrete exponentials**
  - For integers *N, n, r* and *l*

$$\sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}nl} = \begin{cases} N, & l = rN \\ 0, & \text{otherwise} \end{cases}$$

  - In words, summation of harmonically related discrete complex exponentials of $2\pi\ell/N$ is N, if ℓ is an integer multiple of N; zero, otherwise.
  - The proof of the first part follows directly from Euler's expansion, the second part follows from the geometric series expansion:

$$\sum_{n=0}^{N-1}\left(e^{j\frac{2\pi}{N}\ell}\right)^n = \frac{1-\left(e^{j\frac{2\pi}{N}\ell}\right)^N}{1-e^{j\frac{2\pi}{N}\ell}}$$

Note that the exponential in the numerator evaluates to $e^{j2\pi l}$ which is always "1" for any integer "*l*". Therefore, the numerator is zero, and hence the whole expression is zero, except for $l=rN$, when the denominator is zero as well. This undefined situation (0/0) can easily be computed straight from the original expression itself.

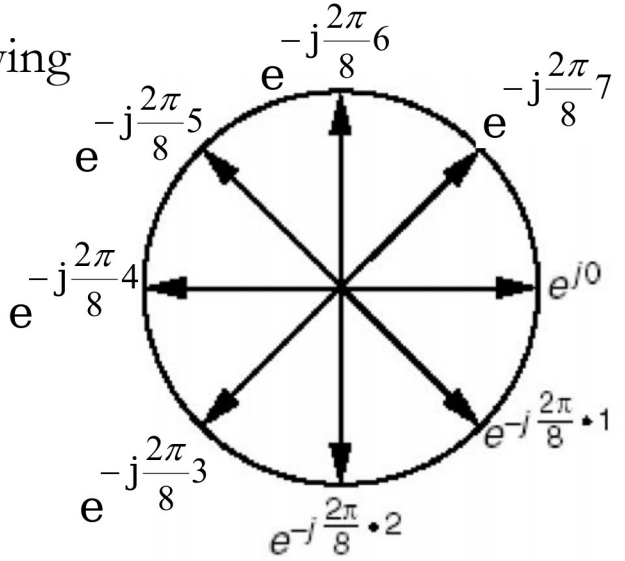


COMPUTING DFT

➲ Recall the analysis equation:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}, \quad 0 \le k \le N-1$$

- For any given ***k***, the DFT is computed by multiplying each x[n] with each of the complex exponentials $e^{j2\pi nk/N}$ and then adding up all these components
- If, for example, we wish to compute an 8 point DFT, the complex exponentials are 8 unit vectors placed at equal distances from each other on the unit circle



$X[0]=x[0]e^{j0}+x[1]e^{j0}+x[2]e^{j0}+x[3]e^{j0}+x[4]e^{j0}+x[5]e^{j0}+x[6]e^{j0}+x[7]e^{j0}$

$X[1]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).1}+x[2]e^{-j(2\pi/8).2}+x[3]e^{-j(2\pi/8).3}+x[4]e^{-j(2\pi/8).4}+x[5]e^{-j(2\pi/8).5}+x[6]e^{-j(2\pi/8).6}+x[7]e^{-j(2\pi/8).7}$

$X[2]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).2}+x[2]e^{-j(2\pi/8).4}+x[3]e^{-j(2\pi/8).6}+x[4]e^{-j(2\pi/8).0}+x[5]e^{-j(2\pi/8).2}+x[6]e^{-j(2\pi/8).4}+x[7]e^{-j(2\pi/8).6}$

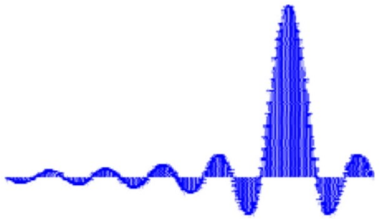
$X[3]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).3}+x[2]e^{-j(2\pi/8).6}+x[3]e^{-j(2\pi/8).1}+x[4]e^{-j(2\pi/8).4}+x[5]e^{-j(2\pi/8).7}+x[6]e^{-j(2\pi/8).2}+x[7]e^{-j(2\pi/8).5}$

$X[4]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).4}+x[2]e^{-j(2\pi/8).0}+x[3]e^{-j(2\pi/8).4}+x[4]e^{-j(2\pi/8).0}+x[5]e^{-j(2\pi/8).4}+x[6]e^{-j(2\pi/8).0}+x[7]e^{-j(2\pi/8).4}$

$X[5]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).5}+x[2]e^{-j(2\pi/8).2}+x[3]e^{-j(2\pi/8).7}+x[4]e^{-j(2\pi/8).4}+x[5]e^{-j(2\pi/8).1}+x[6]e^{-j(2\pi/8).6}+x[7]e^{-j(2\pi/8).3}$

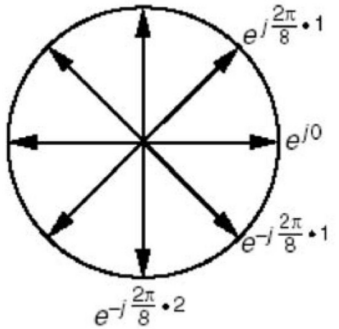
$X[6]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).6}+x[2]e^{-j(2\pi/8).4}+x[3]e^{-j(2\pi/8).2}+x[4]e^{-j(2\pi/8).0}+x[5]e^{-j(2\pi/8).6}+x[6]e^{-j(2\pi/8).4}+x[7]e^{-j(2\pi/8).2}$

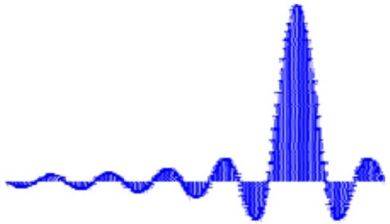
$X[7]=x[0]e^{j0}+x[1]e^{-j(2\pi/8).7}+x[2]e^{-j(2\pi/8).6}+x[3]e^{-j(2\pi/8).5}+x[4]e^{-j(2\pi/8).4}+x[5]e^{-j(2\pi/8).3}+x[6]e^{-j(2\pi/8).2}+x[7]e^{-j(2\pi/8).1}$



Periodicity in DFT

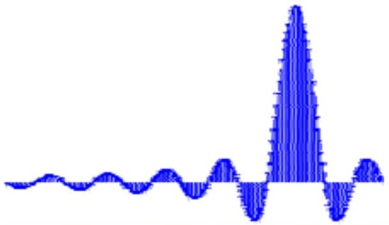
- DFT is periodic in both time and frequency domains!!!
  - Even though the original time domain sequence to be transformed is not periodic!
- There are several ways to explain this phenomenon. Mathematically , we can easily show that both the analysis and synthesis equations are periodic by N.
- Now, understanding that DFT is periodic in frequency domain is straightforward: DFT is obtained by sampling DTFT at $2\pi/N$ intervals. Since DTFT was periodic with 2π → DFT is periodic by N.
- This can also be seen easily from the complex exponential wheel. Since there are only N vectors around a unit circle, the transform will repeat itself every N points.
- But what does it mean that DFT is also periodic in time domain?





Periodicity in DFT

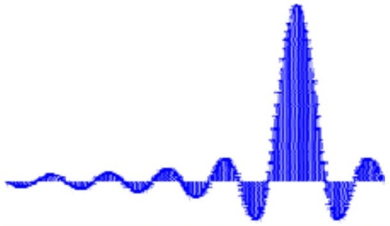
- Recall how we obtained the frequency spectrum of a sampled signal:
  - The spectrum of the sampled signal was identical to that of the continuous signal, except, with periodically replications of itself every 2π.
  - That is, sampling the signal in time domain, caused the frequency domain to be periodic with 2π.
- In obtaining DFT, we did the opposite: sample the frequency domain
  - From the duality of the Fourier transform, this corresponds to making the time domain periodic.
  - However, the original signal was not periodic!
  - This is an artifact of the mathematical manipulations.
- Here is what is happening:
  - **When we sample the DTFT in frequency domain, the resulting "discrete spectrum" is not spectrum of the original discrete signal. Rather, the sampled spectrum is in fact the spectrum of a time domain signal that consists of periodically replicated versions of the original discrete signal.**
  - Similar to sampling theorem, under rather mild conditions, we can reconstruct the DTFT $X(\omega)$, from its DFT samples $X[k]$. More on this later!

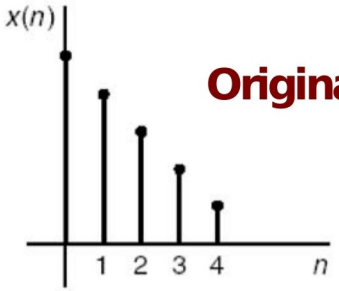


DFT & Circular Shift

- We compute the N-point DFT of a periodic discrete sequence x[n], with the understanding that the computed DFT is in fact the spectrum of a periodic sequence $\tilde{x}[n]$, which is obtained by periodically replicating x[n] with a period of N samples.
- This nuance makes it necessary to introduce the concept of **circular shift**.
  - A circularly shifted sequence is denoted by $x[(n\text{-}L)_N]$, where L is the amount of shift, and N is the length of the previously determined base interval (by default, unless defined otherwise, this is equal to the length of the sequence).
  - The circularly shifted sequence is defined only in the same interval as the original sequence!
  - To obtain a circularly shifted sequence, we first linearly shift the sequence by L, and then rotate the sequence in such a manner that the shifted sequence remain in the same interval originally defined by N.

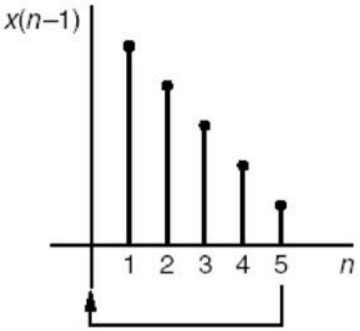
Circular Shift

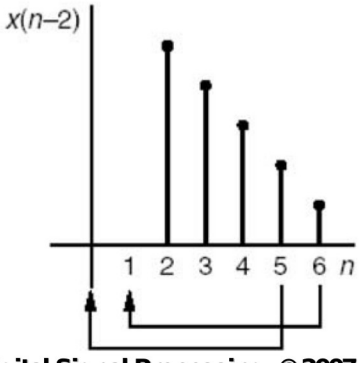




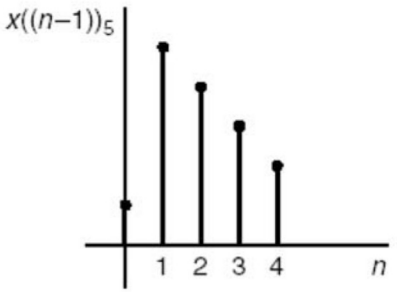
Original sequence

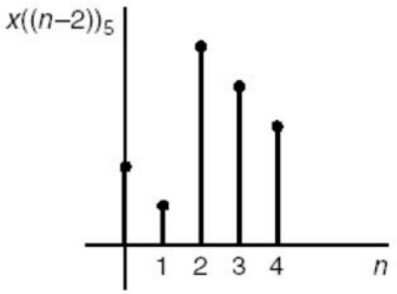
Linear shift

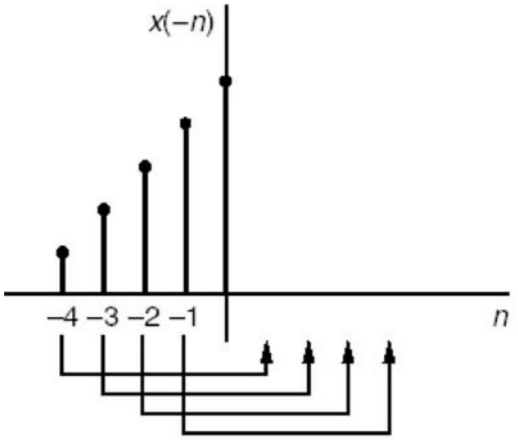


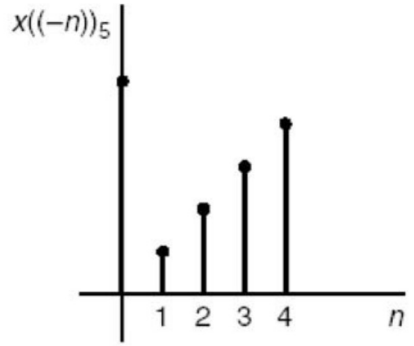


Circular shift





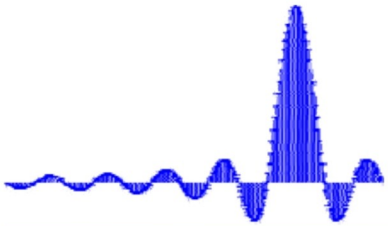




If the original sequence is defined in the time interval N_1 to N_2, then the circular shift can mathematically be represented as follows:

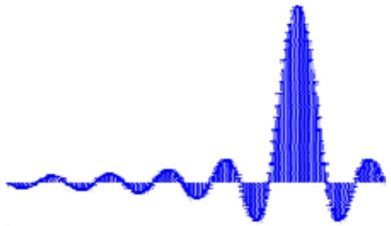
$$\begin{aligned} x_C[n] &= x[(n-L)]_N \\ &= x[(n-L) \bmod N] \\ &= x[(n-L+rN), \text{ such that } N_1 \le n-L \le N_2] \end{aligned}$$

The circularly shifted sequence is obtained by finding an integer **r** such that n-L+rN remains in the same domain as the original sequence.



PROPERTIES OF THE DFT

Type of Property	Length-N Sequence	N-point DFT
	$g[n]$ $h[n]$	$G[k]$ $H[k]$
Linearity	$\alpha g[n] + \beta h[n]$	$\alpha G[k] + \beta H[k]$
Circular time-shifting	$g[\langle n - n_o \rangle_N]$	$W_N^{kn_o} G[k] = e^{-j\frac{2\pi}{N}kn_0} G[k]$
Circular frequency-shifting	$W_N^{-k_o n} g[n] = e^{j\frac{2\pi}{N}k_0 n} g[n]$	$G[\langle k - k_o \rangle_N]$
Duality	$G[n]$	$Ng[\langle -k \rangle_N]$
N-point circular convolution	$\sum_{m=0}^{N-1} g[m]h[\langle n - m \rangle_N]$ $(g[n] * h[n])_N$	$G[k]H[k]$
Modulation (Circular convolution in frequency)	$g[n]h[n]$	$\frac{1}{N}\sum_{m=0}^{N-1} G[m]H[\langle k - m \rangle_N] = \frac{1}{N}(G[k] * H[k])_N$
Parseval's relation	$\sum_{n=0}^{N-1} \lvert x[n] \rvert^2 = \frac{1}{N}\sum_{k=0}^{N-1} \lvert X[k] \rvert^2$	



DFT Symmetry relations

Length-N Sequence	N-point DFT
$x[n]$	$X[k]$
$x^*[n]$	$X^*[\langle -k \rangle_N]$
$x^*[\langle -n \rangle_N]$	$X^*[k]$
$\mathrm{Re}\{x[n]\}$	$X_{\mathrm{pcs}}[k] = \frac{1}{2}\{X[\langle k \rangle_N] + X^*[\langle -k \rangle_N]\}$
$j\,\mathrm{Im}\{x[n]\}$	$X_{\mathrm{pca}}[k] = \frac{1}{2}\{X[\langle k \rangle_N] - X^*[\langle -k \rangle_N]\}$
$x_{\mathrm{pcs}}[n]$	$\mathrm{Re}\{X[k]\}$
$x_{\mathrm{pca}}[n]$	$j\,\mathrm{Im}\{X[k]\}$

Note: $x_{\mathrm{pcs}}[n]$ and $x_{\mathrm{pca}}[n]$ are the periodic conjugate-symmetric and periodic conjugate-antisymmetric parts of $x[n]$, respectively. Likewise, $X_{\mathrm{pcs}}[k]$ and $X_{\mathrm{pca}}[k]$ are the periodic conjugate-symmetric and periodic conjugate-antisymmetric parts of $X[k]$, respectively.

Length-N Sequence	N-point DFT
$x[n]$	$X[k] = \mathrm{Re}\{X[k]\} + j\,\mathrm{Im}\{X[k]\}$
$x_{\mathrm{pe}}[n]$	$\mathrm{Re}\{X[k]\}$
$x_{\mathrm{po}}[n]$	$j\,\mathrm{Im}\{X[k]\}$
Symmetry relations	$X[k] = X^*[\langle -k \rangle_N]$
	$\mathrm{Re}\,X[k] = \mathrm{Re}\,X[\langle -k \rangle_N]$
	$\mathrm{Im}\,X[k] = -\mathrm{Im}\,X[\langle -k \rangle_N]$
	$\lvert X[k] \rvert = \lvert X[\langle -k \rangle_N] \rvert$
	$\arg X[k] = -\arg X[\langle -k \rangle_N]$

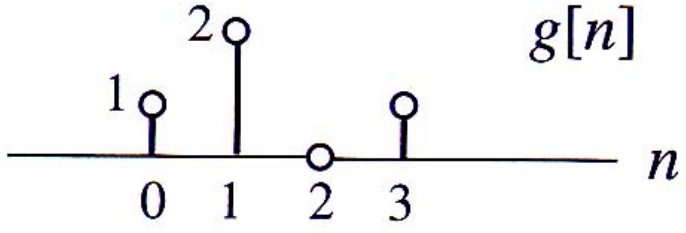
Note: $x_{\mathrm{pe}}[n]$ and $x_{\mathrm{po}}[n]$ are the periodic even and periodic odd parts of $x[n]$, respectively.

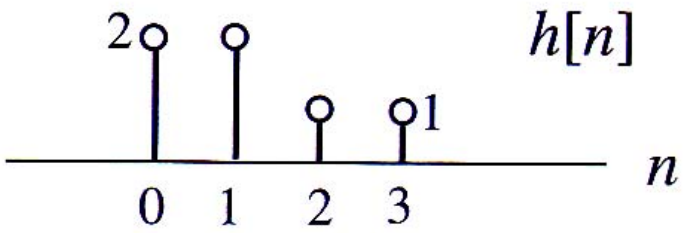
Circular Convolution

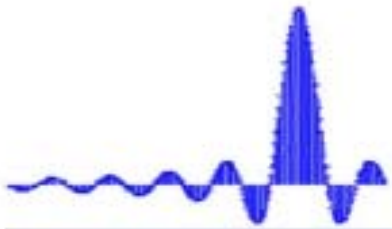
Linear convolution: $$y_L[n] = \sum_{m=0}^{N-1} g[m]h[n-m]$$

Circular convolution: $$y_C[n] = \sum_{m=0}^{N-1} g[m]h[< n-m >_N]$$

consider the following functions for a 4 point circular convolution







Circular Convolution

Evaluating $y_C[n] = \sum_{m=0}^{N-1} g[m]h[< n-m >_N]$ gives:

$$y_C[0] = g[0]h[0] + g[1]h[3] + g[2]h[2] + g[3]h[1]$$
$$y_C[1] = g[0]h[1] + g[1]h[0] + g[2]h[3] + g[3]h[2]$$
$$y_C[2] = g[0]h[2] + g[1]h[1] + g[2]h[0] + g[3]h[3]$$
$$y_C[3] = g[0]h[3] + g[1]h[2] + g[2]h[1] + g[3]h[0]$$

so we see that the 4 terms involve multiplying $g[n]$ with reversed and circularly shifted versions of $h[n]$ on the interval $n = 0 - 3$

Circular Convolution

The reversed and circularly shifted versions of $h[n]$ are:

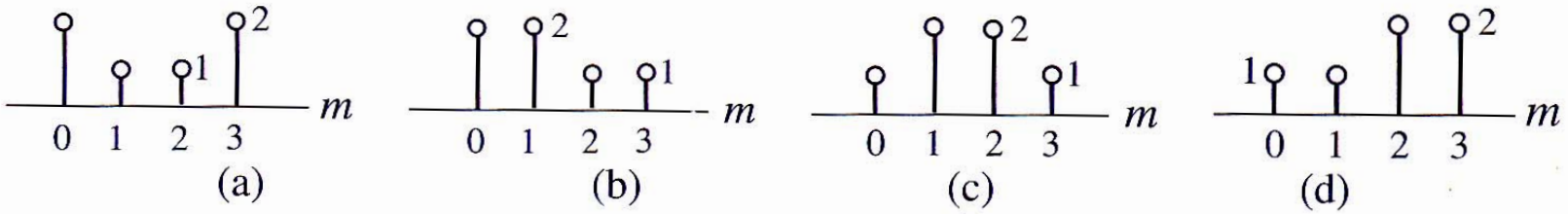
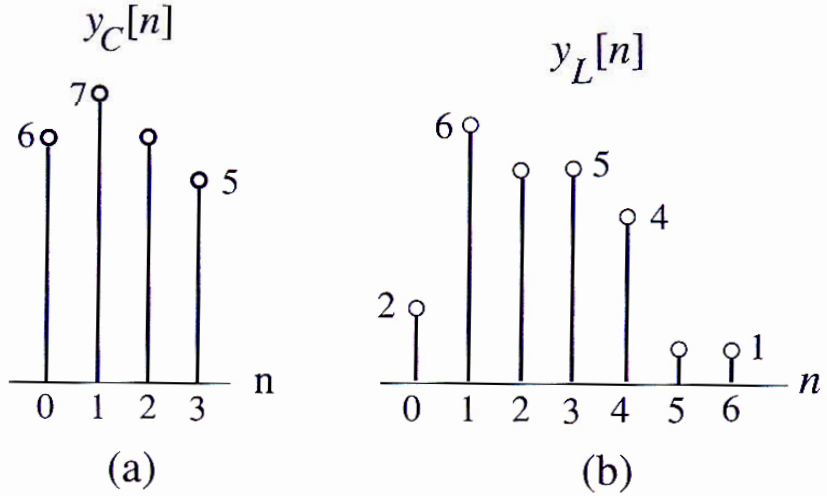


Figure 5.7: The circularly time-reversed sequence and its circularly shifted versions: (a) $h[\langle -m \rangle_4]$, (b) $h[\langle 1-m \rangle_4]$ (c) $h[\langle 2-m \rangle_4]$, and (d) $h[\langle 3-m \rangle_4]$.

leading to the following circular convolution



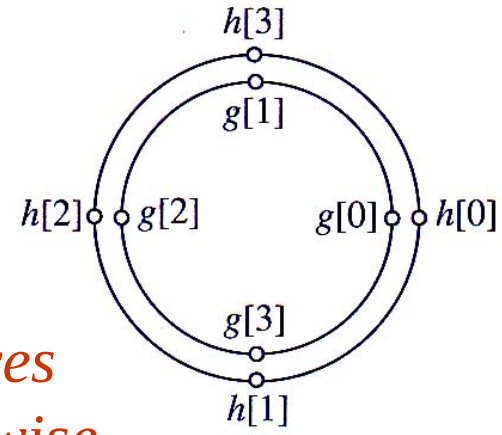
and linear circular convolution results

Figure 5.8: Results of convolution of the two sequences of Figure 5.7. (a) Circular convolution and (b) linear convolution.

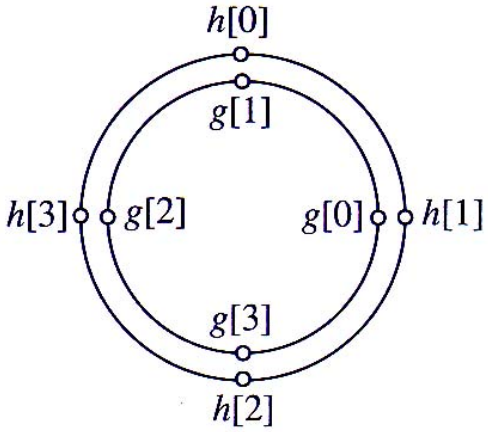
Circular Convolution

A graphical representation of the circular convolution is:

$y[0]$



(a)

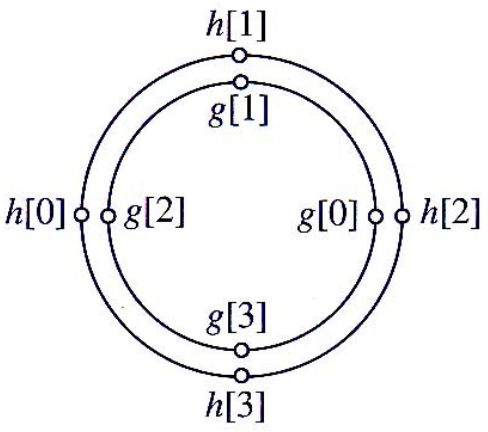


(b)

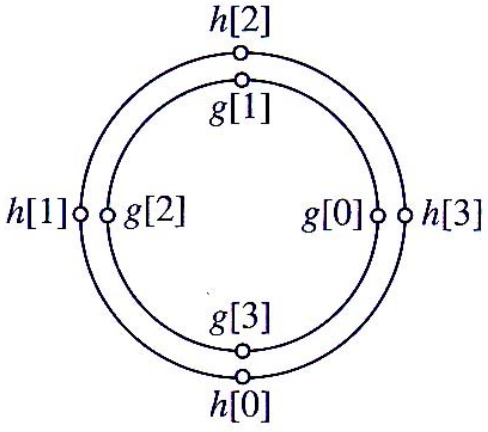
$y[1]$

outer ring moves counter clock wise

$y[2]$

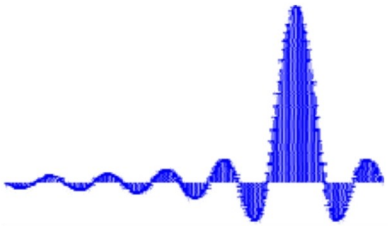


(c)



(d)

$y[3]$

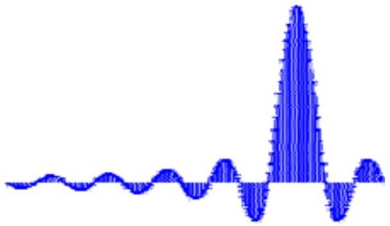


Convolution Matrix

- The circular convolution can also be easily computed using the following **N-point** convolution matrix:

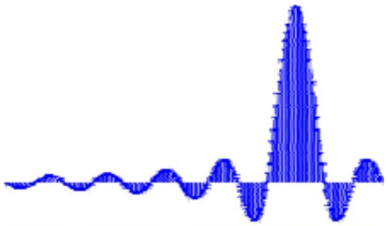
$$y[n] = x[n] \circledast_N h[n]$$

$$\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[N-1] \end{bmatrix} = \begin{bmatrix} h[0] & h[N-1] & h[N-2] & \cdots & h[1] \\ h[1] & h[0] & h[N-1] & \cdots & h[2] \\ h[2] & h[1] & h[0] & \cdots & h[3] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h[N-1] & h[N-2] & h[N-3] & \cdots & h[0] \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$



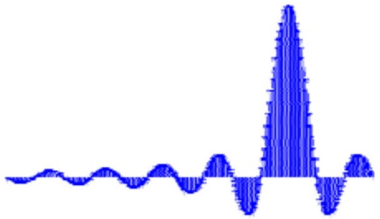
Linear vs. Circular Convolution

- Note that the results of linear and circular convolution are different. This is a problem! Why?
- All LTI systems are based on the principle of linear convolution, as the output of an LTI system is the linear convolution of the system impulse response and the input to the system, which is equivalent to the product of the respective DTFTs in the frequency domain.
  - However, if we use DFT instead of DTFT (so that we can compute it using a computer), then the result appear to be invalid:
    - DTFT is based on linear convolution, and DFT is based on circular convolution, and they are not the same!!!
    - For starters, they are not even of equal length: For two sequences of length N and M, the linear convolution is of length N+M-1, whereas circular convolution of the same two sequences is of length max(N,M), where the shorter sequence is zero padded to make it the same length as the longer one.
    - Is there any relationship between the linear and circular convolutions? Can one be obtained from the other? OR can they be made equivalent?



Linear vs. Circular Convolution

- YES!, rather easily, as a matter of fact!
  - **FACT:** If we ***zero pad*** both sequences x[n] and h[n], so that they are both of length N+M-1, then linear convolution and circular convolution result in identical sequences
  - **Furthermore:** If the respective DFTs of the zero padded sequences are X[k] and H[k], then the inverse DFT of X[k]·H[k] is equal to the linear convolution of x[n] and h[n]
  - Note that, normally, the inverse DFT of X[k].H[k] is the circular convolution of x[n] and h[n]. If they are zero padded, then the inverse DFT is the linear convolution of the two.



EXAMPLE

- Compute circular convolution of x[n]=[**1** 3 2 -1 4], h[n]=[**2** 0 1 7 -3], by appropriately zero padding the two

Zero pad signals!

Corresponding convolution formula

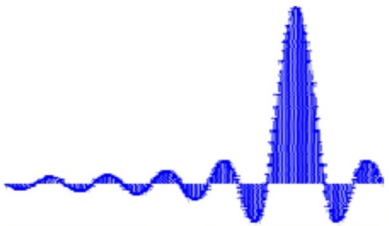
Solution

Computing Convolution Using DFT

- Note that we can compute the convolution using no time domain operation, using the convolution property of DFT, that is, the inverse DFT of X[k]·H[k] is equal to the linear convolution of x[n] and h[n].
- Since we can compute DFT very efficiently using FFT (more on this later), it is actually more computationally efficient to compute the DFTs X[k] and H[k], multiply them, and take the inverse DFT then to compute the convolution in time domain:

$$y[n] = x[n] \otimes h[n] = IDFT\{X[k].H[k]\}$$

- Note, however, the IDFT gives the circular convolution. **To ensure that one gets linear convolution, both sequences in the time domain must be zero padded to appropriate length**

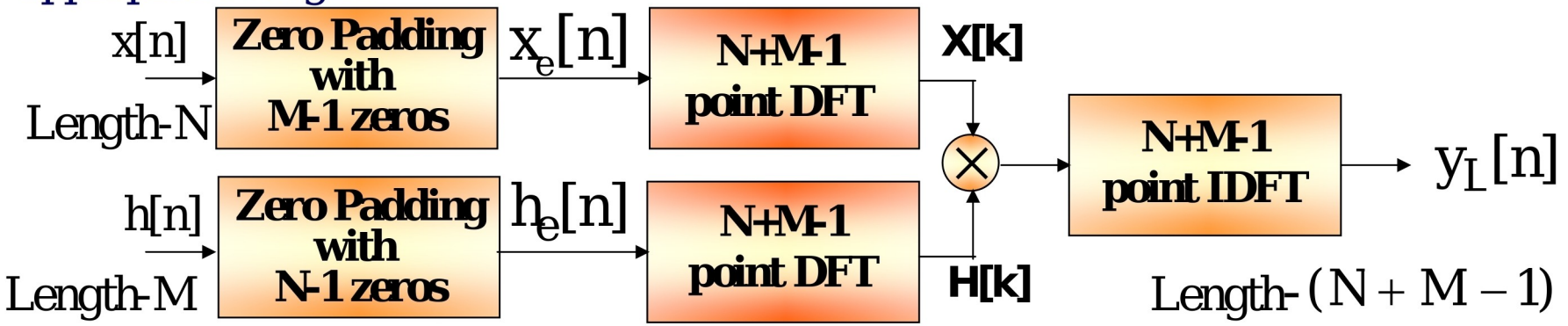


Computing Convolution Using DFT

- Note that we can compute the convolution using no time domain operation, using the convolution property of DFT, that is, the inverse DFT of X[k]·H[k] is equal to the linear convolution of x[n] and h[n].
- Since we can compute DFT very efficiently using FFT (more on this later), it is actually more computationally efficient to compute the DFTs X[k] and H[k], multiply them, and take the inverse DFT then to compute the convolution in time domain:

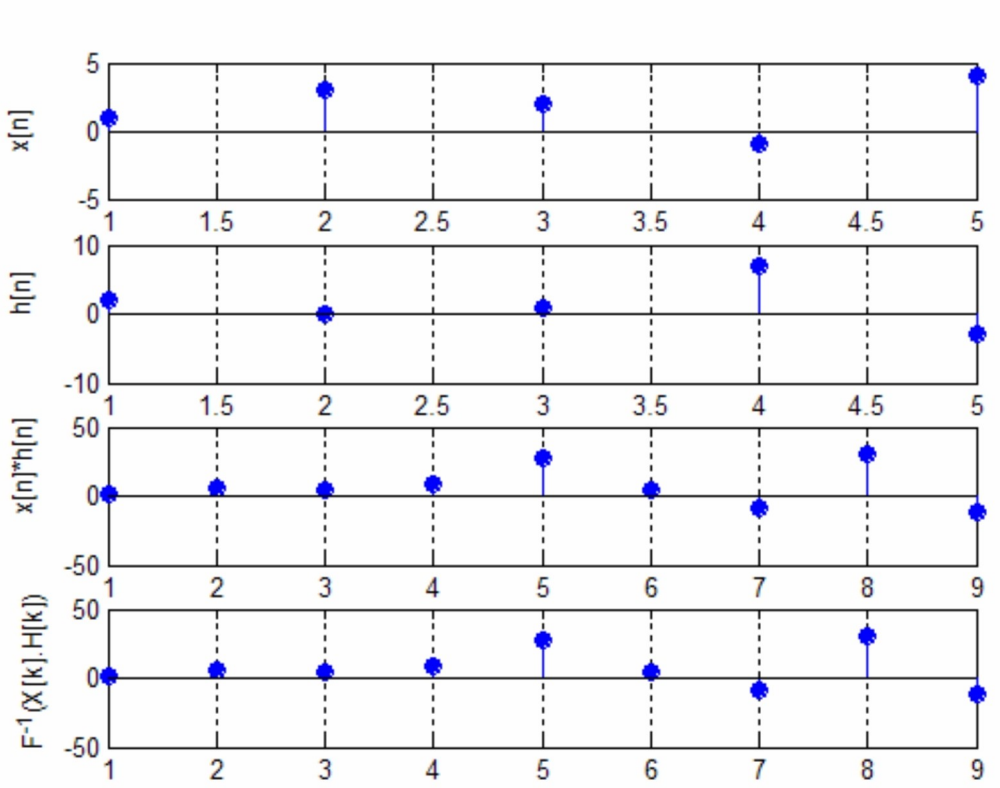
$$y[n] = x[n] \circledast_N h[n] = \left(x[n] * h[n] \right)_N = \text{IDFT}\{X[k] \cdot H[k]\}$$

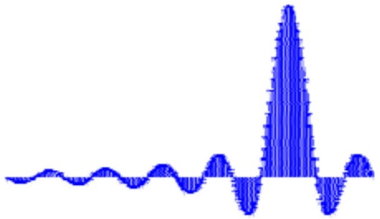
- Note, however, the IDFT gives the circular convolution. To ensure that one gets linear convolution, both sequences in the time domain must be zero padded to appropriate length.



EXAMPLE

```
x=[1 3 2 -1 4];
h=[2 0 1 7 -3];
x2=[1 3 2 -1 4 0 0 0 0 0];
h2=[2 0 1 7 -3 0 0 0 0 0];
C1=conv(x,h);
X=fft(x2); H=fft(h2);
C2=ifft(X.*H);
subplot(411)
stem(x, 'filled'); grid
subplot(412)
stem(h, 'filled'); grid
subplot(413)
stem(C1, 'filled'); grid
subplot(414)
stem(real(C2), 'filled');  grid
```





IN MATLAB

- In Matlab, the **fft()** computes DFT using a fast algorithm, called **fast Fourier transform (FFT)**.
- X = fft(x) returns the discrete Fourier transform (DFT) of vector X, computed with a fast Fourier transform (FFT) algorithm.
  - If x is a matrix, **fft** returns the Fourier transform of each column of the matrix. In this case, the length of X and the length of x are identical.
  - X = fft(x,N) returns the N-point DFT. If the length of X is less than N, X is padded with trailing zeros to length N. If the length of X is greater than N, the sequence X is truncated.
  - The N points returned by fft corresponds to frequencies in the [0 2π] range, equally spaced with an interval of $2\pi/N$.
  - Note that the N^{th} FFT point corresponds to 2π, which in turn corresponds to the sampling frequency.
  - If x[n] is real, X[k] is symmetric. Using the fftshift() function shifts the center of symmetry so that the FFT is given in the [$-\pi$ to π] interval, rather then [0 2π].
- X=ifft(X,N) returns the N-point inverse discrete Fourier transform

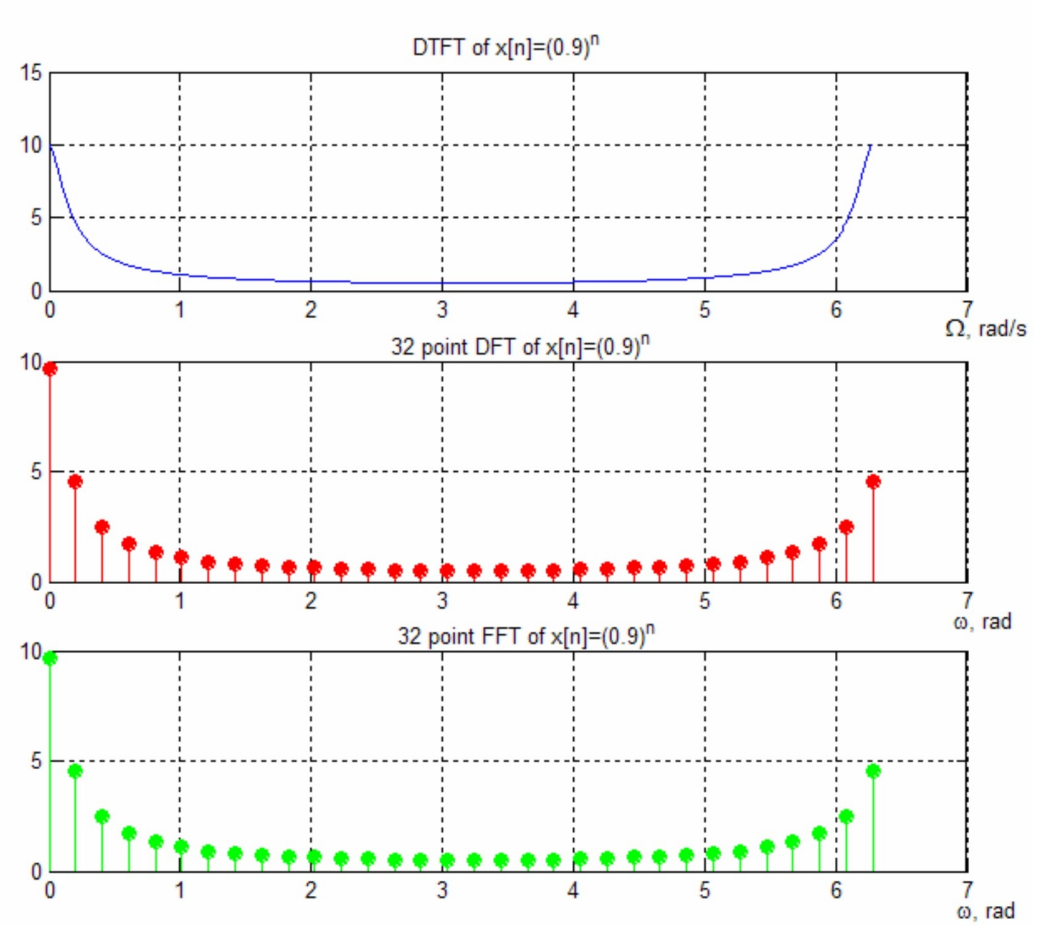
Back to Example

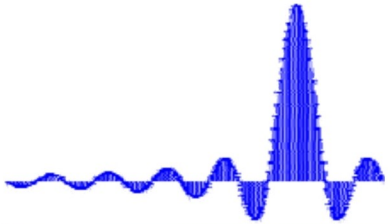
Let's show that

i. DFT is indeed the sampled version of DTFT and

ii. DFT and FFT produce identical results

```
n=0:31; k=0:31;
x=0.9.^n;
w=linspace(0, 2*pi, 512);
K=linspace(0, 2*pi, 32);
X1=1./(1-0.9*exp(-j*w));
X2=(1-(0.9*exp(-j*(2*pi/32)*k)).^32)./
(1-0.9*exp(-j*(2*pi/32)*k));
X=fft(x);
subplot(311)
plot(w, abs(X1)); grid
subplot(312)
stem(K, abs(X2), 'r', 'filled'); grid
subplot(313)
stem(K, abs(X), 'g', 'filled'); grid
```





Matrix Computation of DFT

- DFT has a simple matrix implementation: The DFT samples defined by

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

can be expressed in a matrix form $\mathbf{X} = \mathbf{D}_N \mathbf{x}$

where $\mathbf{X}=[X[0]\ X[1]\ \ldots\ X[N-1]]^T$ is the vectorized DFT sequence,
$\mathbf{x}=[x[0]\ x[1]\ \ldots\ x[N-1]]^T$ is the vectorized time domain sequence, and
$\mathbf{D}_N$ is the NxN **DFT matrix** given by

$$\mathbf{D}_N = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N^1 & W_N^2 & \cdots & W_N^{(N-1)} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & W_N^{(N-1)} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)^2} \end{bmatrix}$$

The Fast Fourier Transform (FFT)

Chapter 11 Mitra

- By far one of the most influential algorithms ever developed in signal processing, revolutionalizing the field
- FFT: Computationally efficient calculation of the frequency spectrum
  - Made many advances in signal processing possible
  - **Drastically** reduces the number of additions and multiplications necessary to compute the DFT
- Many competing algorithms
  - Decimation in time FFT
  - Decimation in frequency FFT

Recall: $W_N = e^{-j2\pi/N}$

- Makes strategic use of the two simple complex identities:

$$W_N^2 = e^{-j\frac{2\pi}{N}2} = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$$

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

$$W_N^{N/2} = -1$$

$$W_N^{N/4} = j$$

$$W_N^{3N/4} = -j$$

$$W_N^0 = 1$$

Computational Complexity of DFT

- Q: How many multiplications and additions are needed to compute DFT?

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

- Note that for each "k" we need N complex multiplications, and N-1 complex additions **(W_N^{kn}) does not depend on x[n], and hence can be precomputed and saved in table**
  - Each complex multiplication is four real multiplications and two real additions
  - A complex addition requires two real additions
  - So for N values of "k", we need N*N complex multiplications and N*(N-1) complex additions
    - This amounts to N^2 complex multiplications and N*(N-1)~N^2 (for large N) complex additions
    - N^2 complex multiplications: $4N^2$ real multiplications and ~$2N^2$ real additions
    - N^2 complex additions: $2N^2$ real additions
  - A grand total of $4N^2$ real multiplications and $4N^2$ real additions:
    - The computational complexity grows with the square of the signal size.
    - This computational complexity is referred to as $O(N^2)$, also called, order of N^2
    - For , say 1000 point signal: 4,000,000 multiplications and 4,000,000 additions: OUCH!

FFT: Decimation in Time

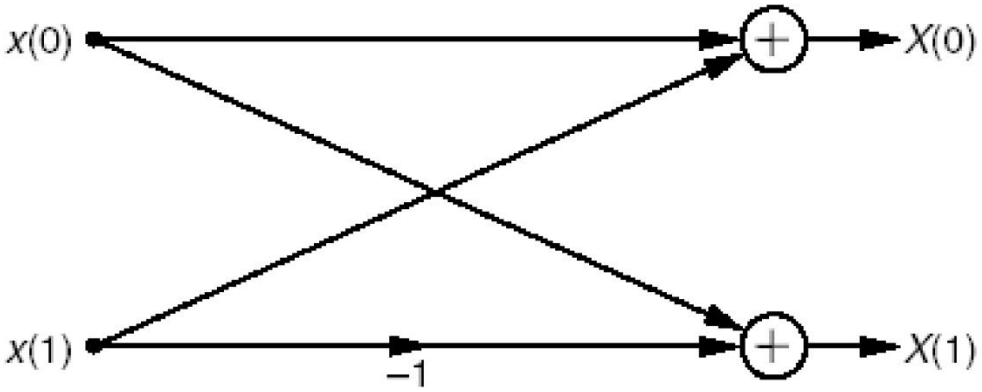
- Assume that the signal is of length $N=2^p$, a power of two. If it is not, zero-pad the signal with enough number of zeros to ensure power-of-two length.
- N-point DFT can be computed as two N/2 point DFTs, both of which can then be computed as two N/4 point DFTs
  - Therefore an N-point DFT can be computed as four N/4 point DFTs.
  - Similarly, an N/4 point DFT can be computed as two N/8 point DFTs ➔ the entire N-point DFT can then be computed as eight N/8 point DFTs
  - Continuing in this fashion, an N-point DFT can be computed as N/2 2-point DFTs
    - A two point DFT requires no multiplications (!) and just two additions.
    - We will see that we will need additional multiplications and additions to combine the 2-point DFTs, however, overall, the total number of operations will be much fewer than that would be required by the regular computation of DFT.

TWO-POINT DFT

- How many operations do we need for 2-point DFT?
  - X[0]=x[0]+x[1]
  - X[1]=x[0]*1+x[1]*$e^{-j\pi}$=x[0]-x[1]

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

$$W_N = e^{-j2\pi/N}$$



2-point DFT requires no multiplication, just two additions

(sign change is very inexpensive – reversal of one or more bits – and hence does not even go into complexity computations)

Decimation in Time

- For the first stage, we first "decimate" the time-domain signal into two half: even indexed samples and odd indexed samples.
  - Using the first property of complex exponentials:

$$X[k] = \underbrace{\sum_{n=0}^{(N/2)-1} x[2n] W_{(N/2)}^{kn}}_{P[k]} + W_N^k \underbrace{\sum_{n=0}^{(N/2)-1} x[2n+1] W_{(N/2)}^{kn}}_{S[k]}, \quad 0 \le k \le \frac{N}{2} - 1$$

N/2 point DFT of even samples

N/2 point DFT of odd samples – defined on k=[0, 1, …, (N/2)-1]

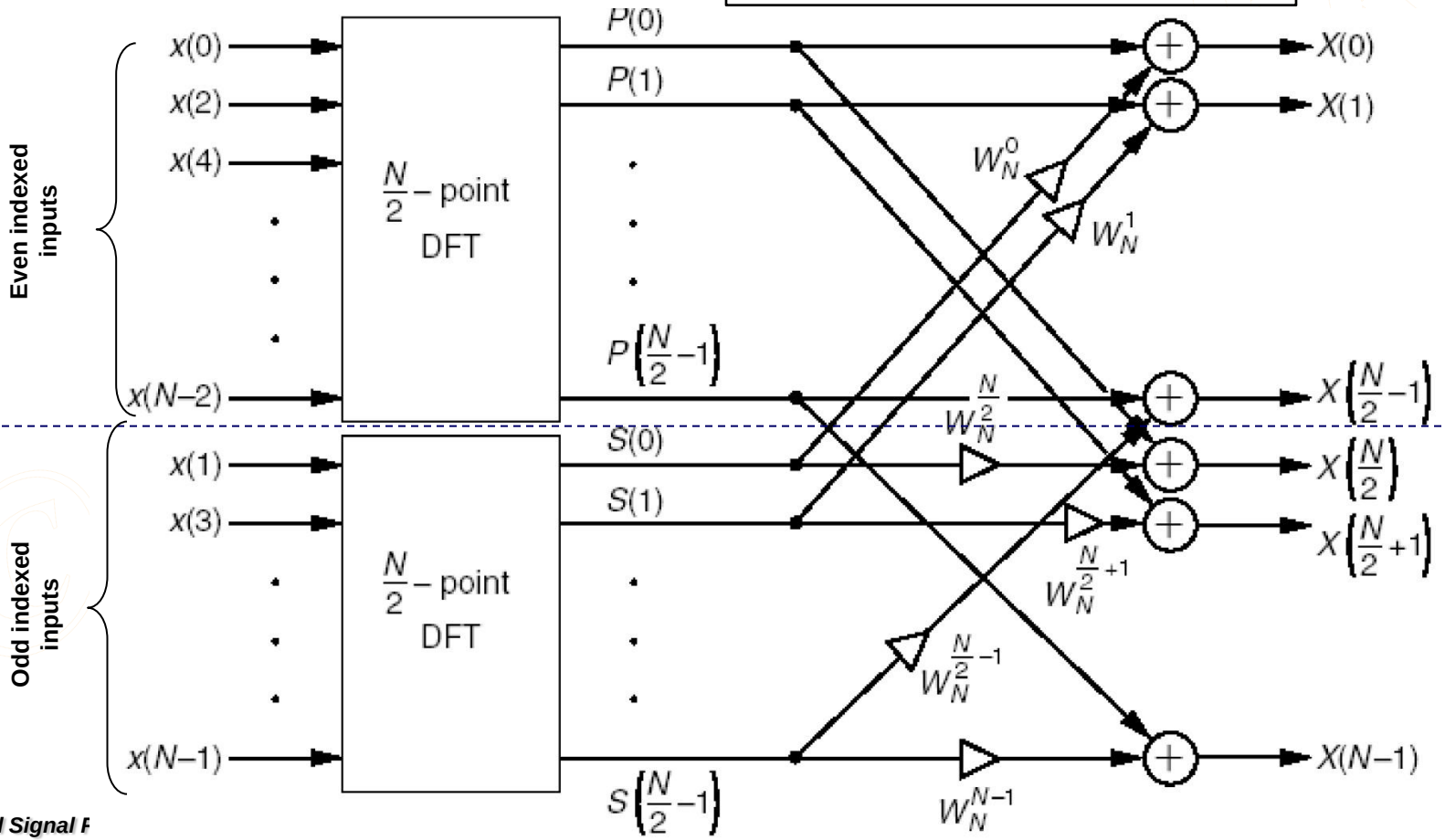
  - However, we need all values of P[k] and S[k] for k=[0~N-1]
  - Since DFT for real sequences is even, P[k] and S[k] are identical in both [0 (N/2)-1] and [(N/2) N-1] intervals ➔ P[k]=P[k+N/2], and similarly, S[k]=S[k+N/2]

Decimation in Time Stage 1

➲ Then: $X[k] = P[k] + W_N^k S[k] \quad 0 \le k \le \frac{N}{2} - 1$

$$W_N^{k+N/2} = W_N^k \cdot W_N^{N/2} = -W_N^k$$

$$X[k + \frac{N}{2}] = P[k + \frac{N}{2}] + W_N^{k+N/2} S[k + \frac{N}{2}]$$

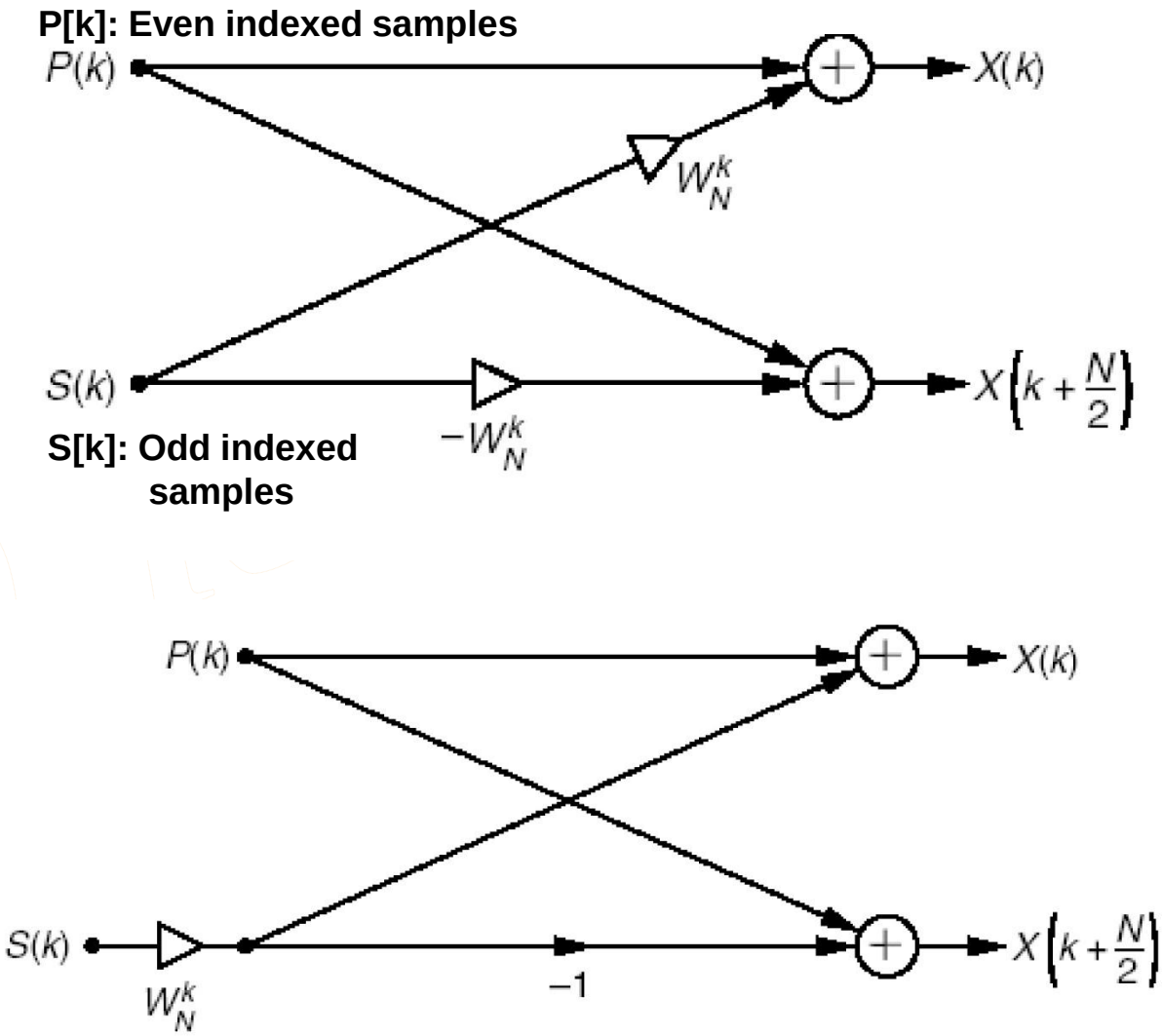


k=0~(N/2)-1

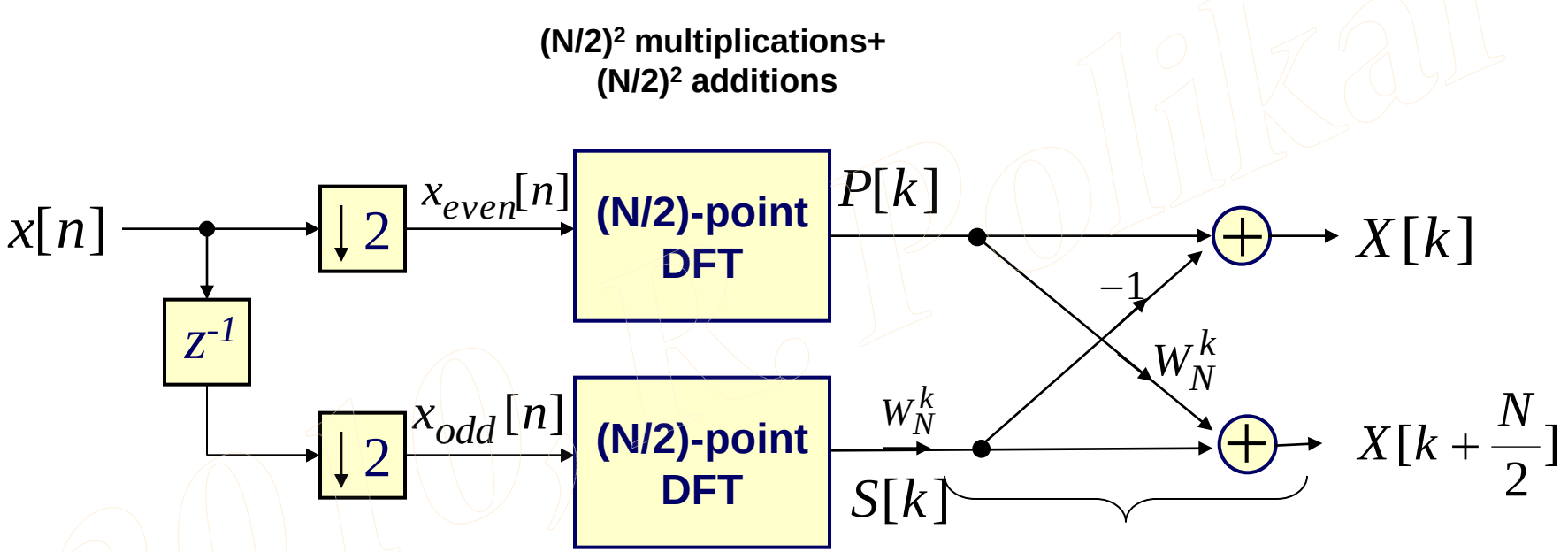
k=(N/2)~N-1

The Butterfly Process

- The basic building block in this implementation can be represented by the ***butterfly process.***
  - Two multiplications & two additions
- But we can do even better, just by moving the multiplier before the node S[k]
  - Reduced butterfly
  - 1 multiplication and two additions !
- We need N/2 reduced butterflies in stage 1 to connect the even and odd N/2 point DFTs
  - A total of (N/2) multiplications and 2*(N/2)=N additions

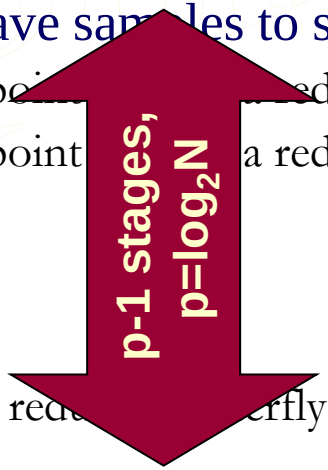


Stage 1 FFT

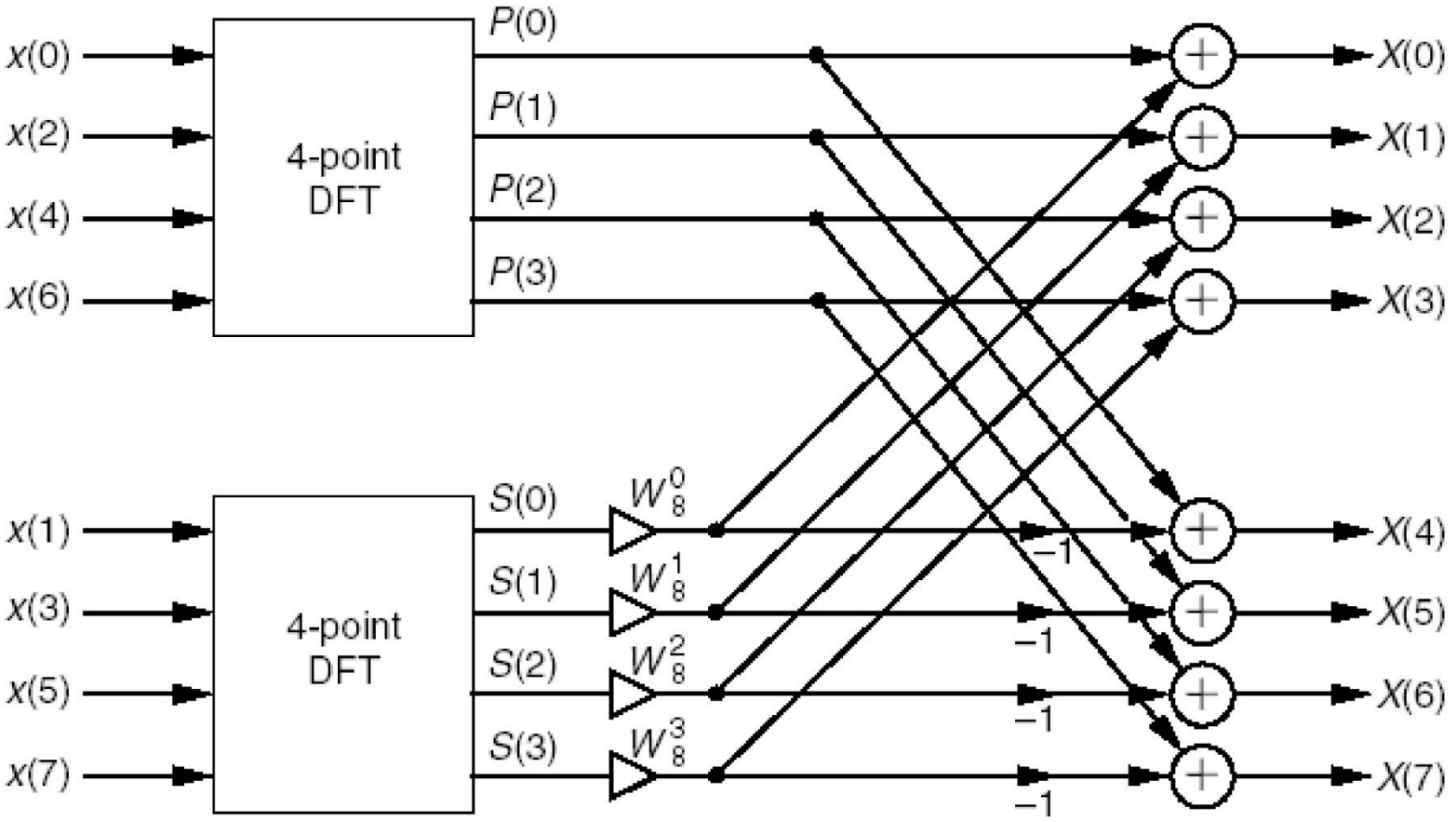


Stage 2…& 3…&4…

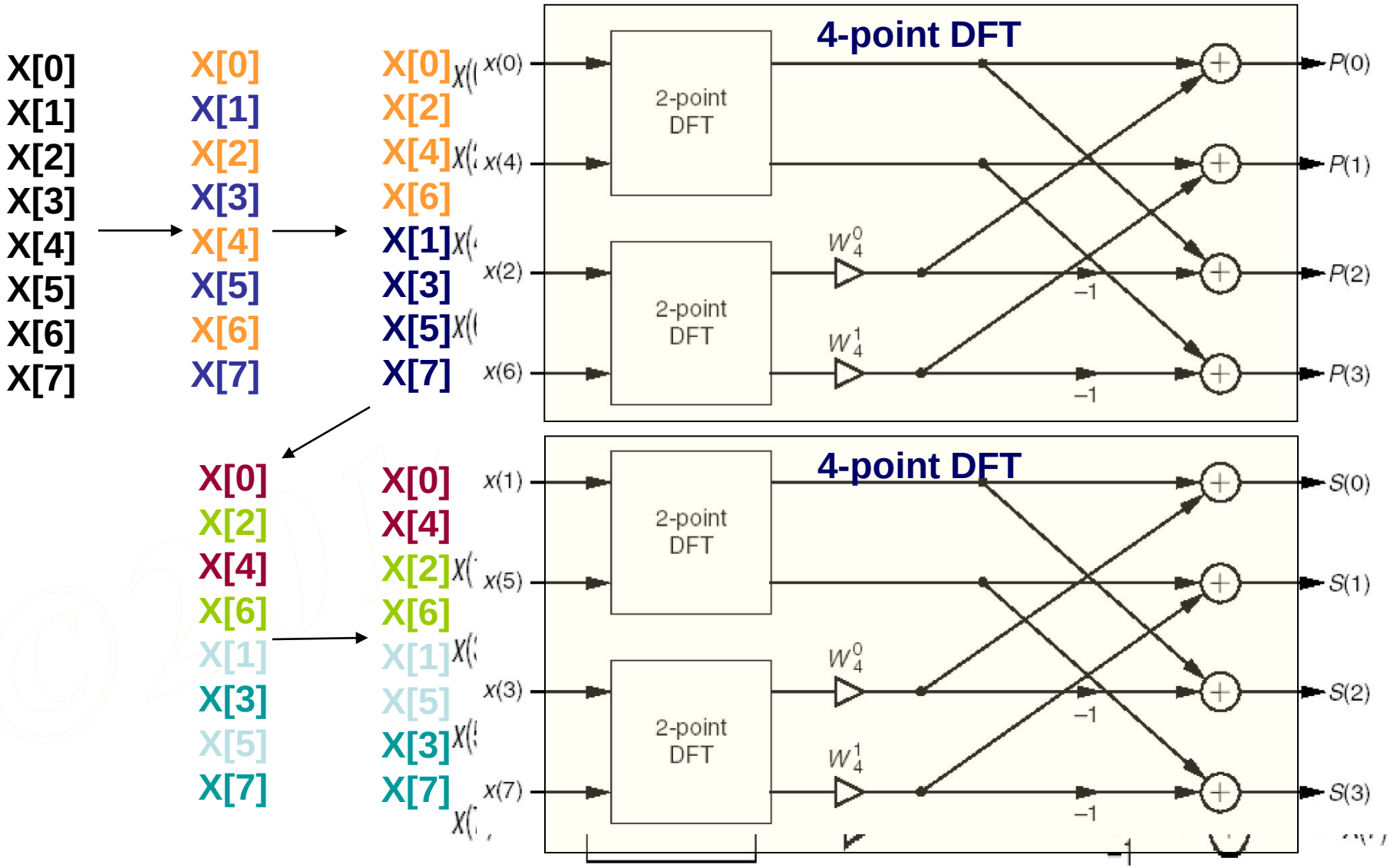
- But wait…!
- We can continue the same process as long as we have samples to split in half:
  - Each N/2-point DFT can be computed as two N/4-point … reduced butterfly
  - Each N/4-point DFT can be computed as two N/8-point … a reduced butterfly.
  - …
  - …
  - …
  - 2 2-point DFT (which requires no multiplications)+ a red… erfly



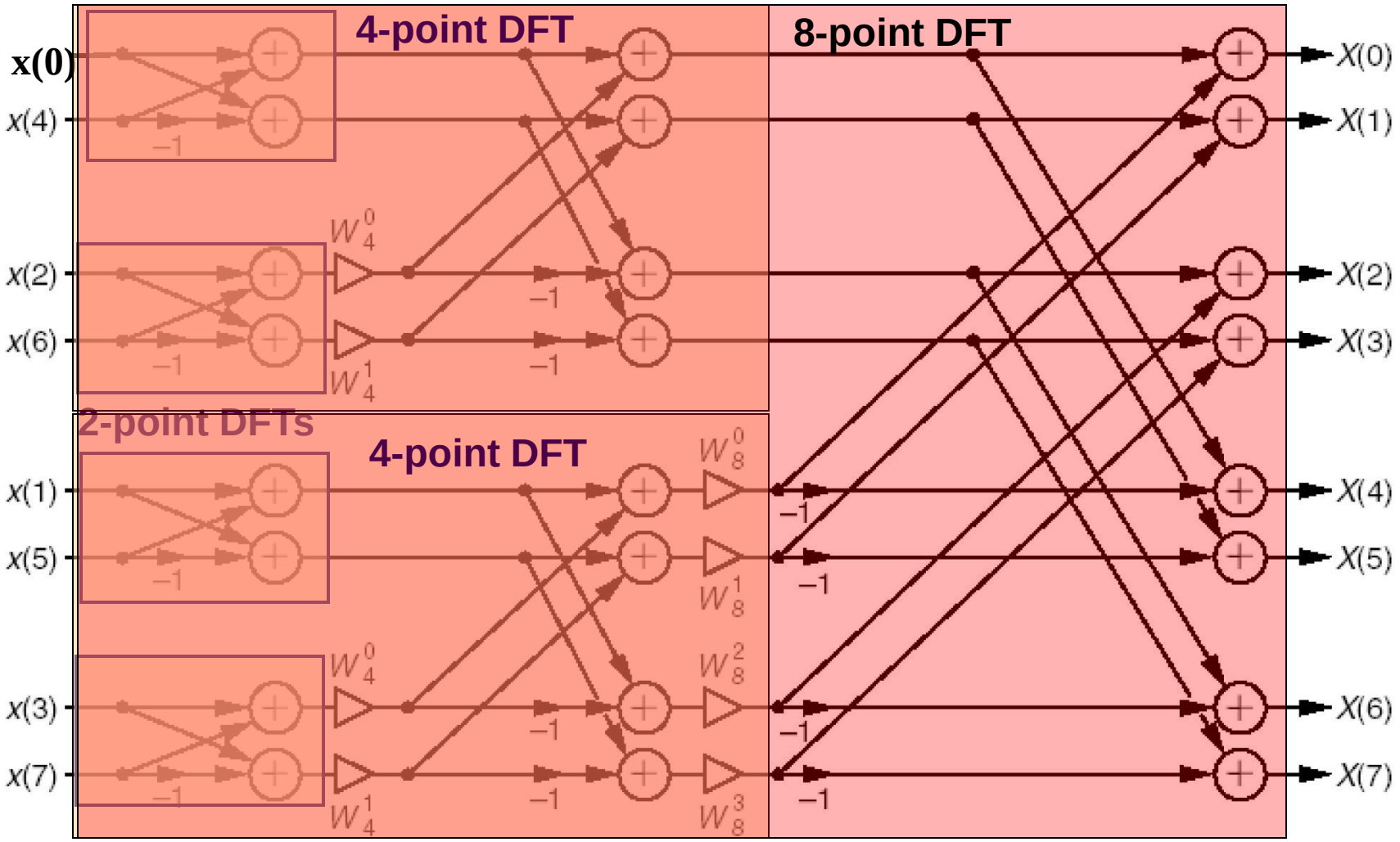
8-point FFT – Stage 1



8-Point FFT – Stage 2



8-Point FFT – Stage 3



Note that further reductions can be obtained by avoiding the following multiplications

$$W_N^{N/2} = -1 \qquad W_N^{N/4} = j \qquad W_N^{3N/4} = -j \qquad W_N^0 = 1$$

So How did we do?

In $\log_2 N$ stages, total # of calculations:

$(N/2)*(\log_2 N-1) \sim (N/2)*\log_2 N$ multiplications

$N\log_2 N$ additions

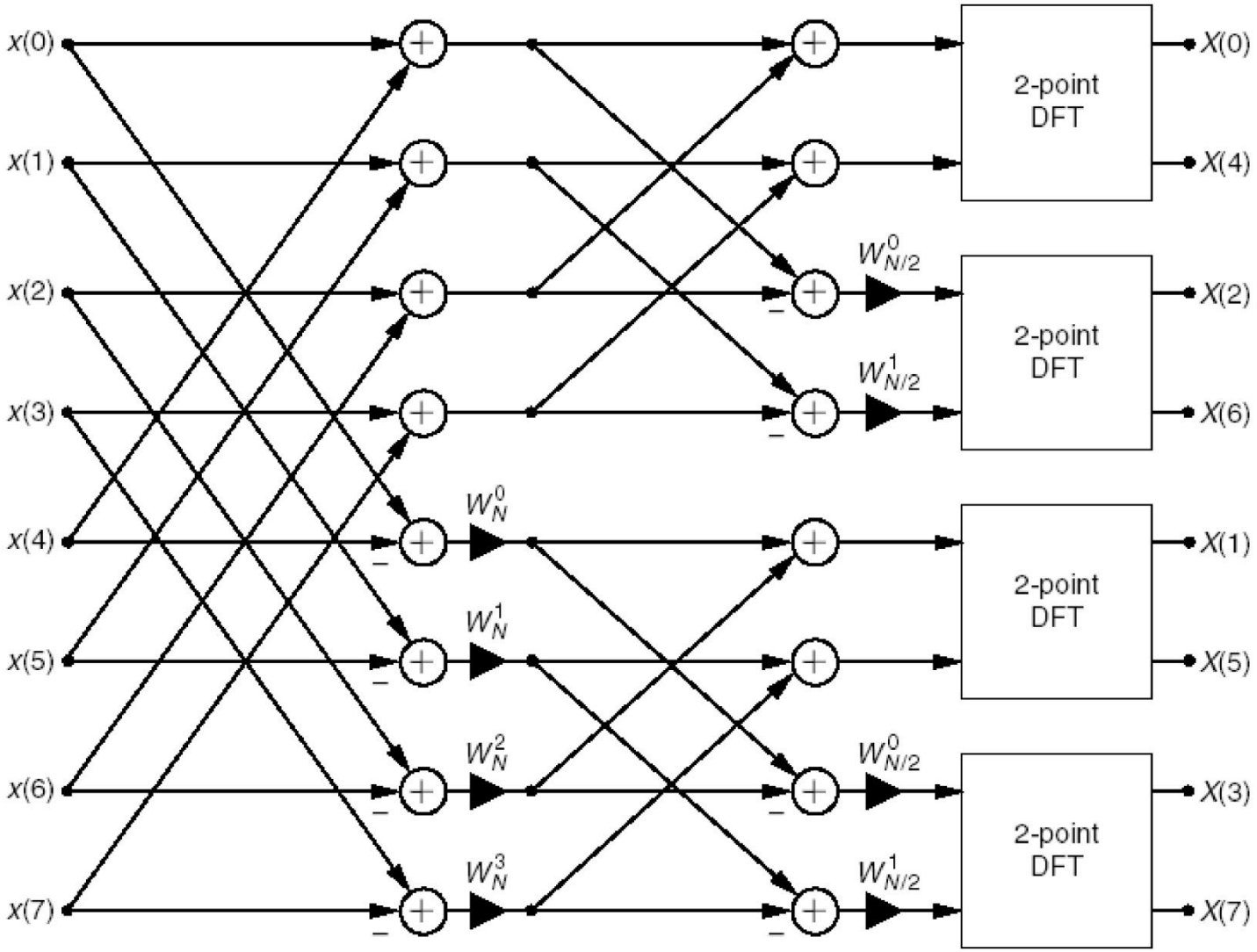
As opposed to N^2 multiplications and N^2 additions

Is that good…?

Decimation in Frequency

- Just like we started with time-domain samples and decimated them in half, we can do the same thing with frequency samples:
  - Compute the even indexed spectral samples k=0, 2, 4,…, 2N-1 and odd indexed spectral components k=1, 3, 5, …, 2N-2 separately.
  - Then further divide each by half to compute X[k] for 0, 4, 8,… and 2, 6, 10, … as well as 1, 5, 9, … and 3, 7, 11 …separately
  - Continuing in this manner, we can reduce the entire process to a series of 2-point DFTs.

8-point DFT with Decimation in Frequency



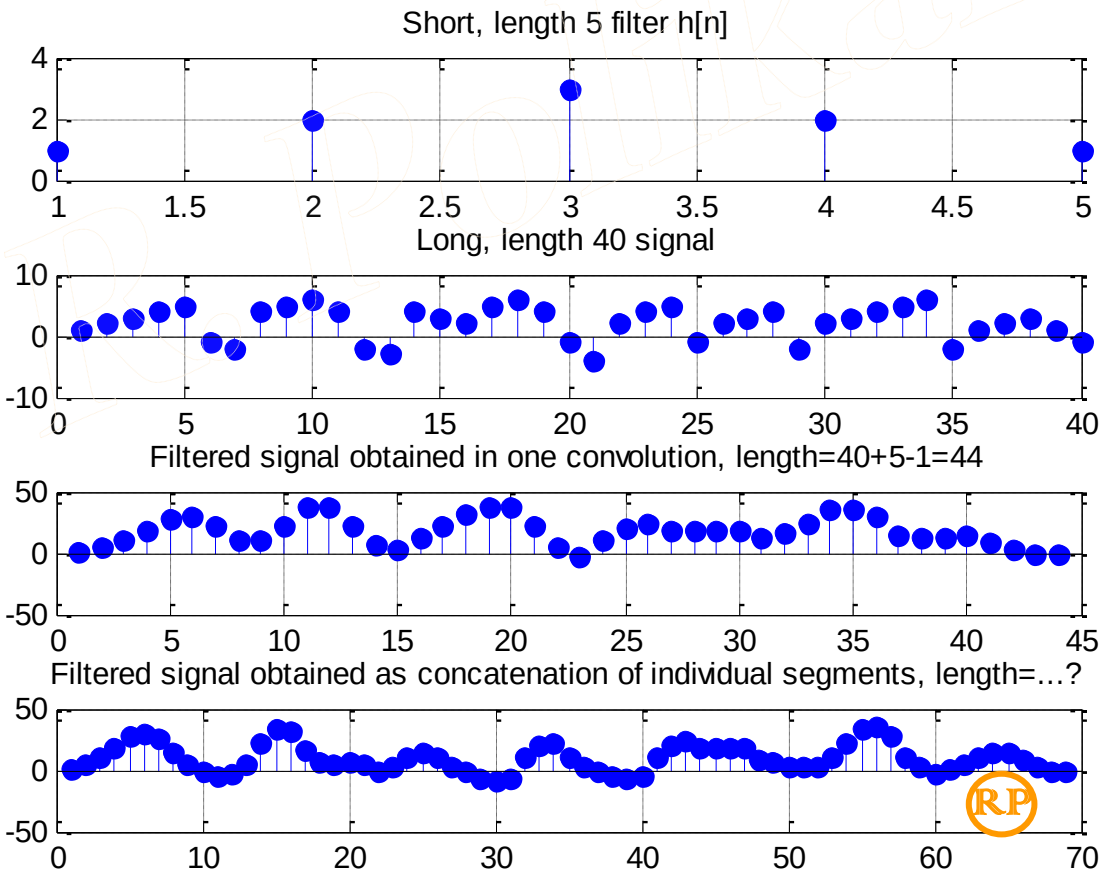
Filtering Streaming Data

- In most real-world applications, the filter length is actually rather small (typically, N<100); however, the input signal is obtained as a streaming data, and therefore can be very long.
- To calculate the output of the filter, we can
  a) Wait until we receive all the data, and then do a full linear convolution of length N filter and length M x[n], where M>>>>N ➔ **y[n]=x[n]*h[n]**
  b) We can use the DFT based method ➔ **y[n]=IDFT(X[k].H[k])**, but we still need to wait for the entire data to arrive to calculate **X[k]**
- In either case, the calculation is very long, expensive, and need to wait for the entire data to arrive.
- Can we just process the data in batches, say 1000 samples at a time, and then concatenate the results…?

What Happened...?

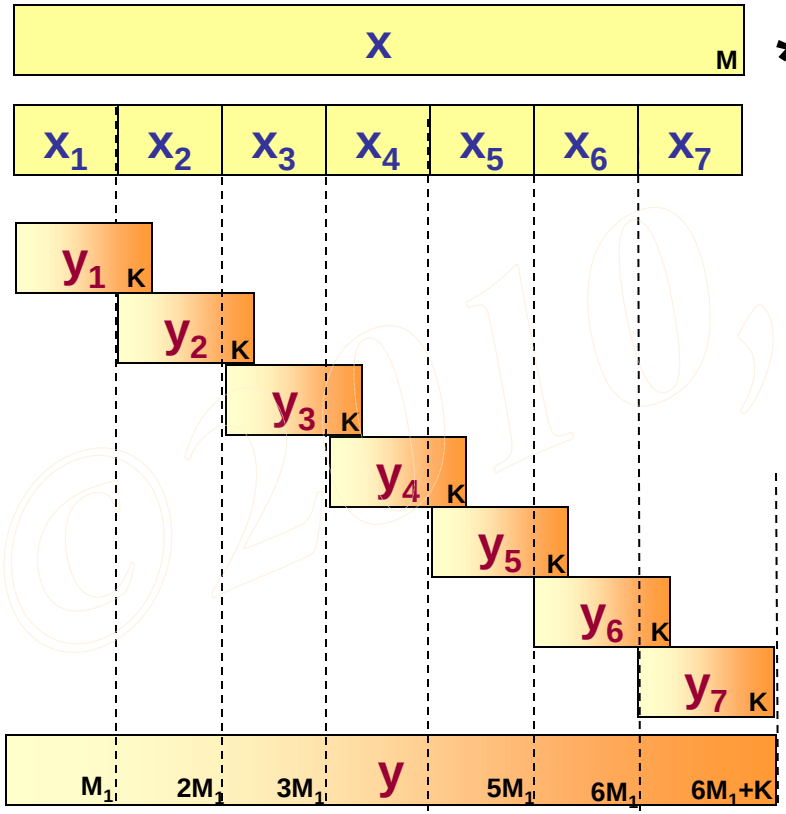
- An example:
  - h[n]=[1 2 3 2 1]; (a short filter)
  - x[n]=[1 2 3 4 5 -1 -2 4 5 6 4 -2 -3 4 3 2 5 6 4 -1 -4 2 4 5 -1 2 3 4 -2 2 3 4 5 6 -2 1 2 3 1 -1]

```
h=[1 2 3 2 1]; % Some short filter N=5;
x=[1 2 3 4 5 -1 -2 4 5 6 4 -2 -3 4 3 2 5 6 4 -1 -4 ...
   2 4 5 -1 2 3 4 -2 2 3 4 5 6 -2 1 2 3 1 -1]; % M=40
y=conv(x, h); % regular convolution Length N+M-1
%Let's divide the long x[n] into smaller segments and
calculate the convolution of each segment
x1=x(1:5);    y1=conv(x1, h);
x2=x(6:10);   y2=conv(x2, h);
x3=x(11:15);  y3=conv(x3, h);
x4=x(20:25);  y4=conv(x4, h);
x5=x(21:30);  y5=conv(x5, h);
x6=x(31:35);  y6=conv(x6, h);
x7=x(36:40);  y7=conv(x7, h);
% Now add those segments together
y_batch=[y1 y2 y3 y4 y5 y6 y7];
%and compare to the original convolution
subplot(411); stem(h, 'filled'); grid
title('Short, length 5 filter h[n]')
subplot(412); stem(x, 'filled'); grid
title('Long, length 40 signal')
subplot(413); stem(y, 'filled'); grid
title('Single convolution, length=40+5-1=44')
subplot(414); stem(y_batch, 'filled'); grid
title('Concatenation of individual segments, length=...?')
```



Overlap Add

- The border effect! Processing individual segments separately and then combining them is possible, however, the border distortion needs to be addressed
  - Overlapp add is a method that allows us to compute the individual segments and then concatenate them in such a way that the concatenated signal is the same as the one that would be obtained if we processed the entire data at once.



$M = 7M_1$

$K = M_1 + N - 1$

$6M_1 + K =$

$= 6M_1 + M_1 + N - 1$

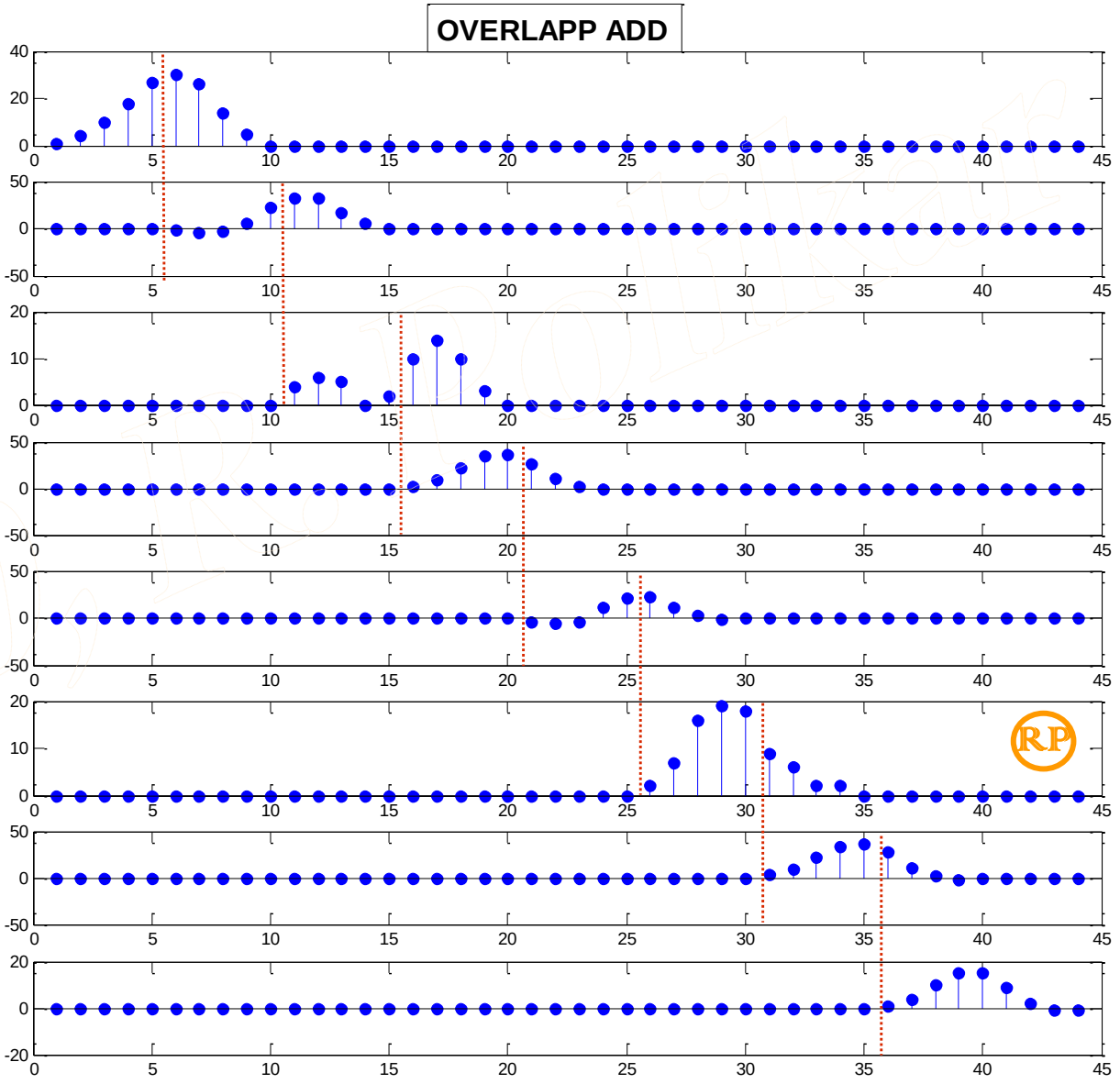
$= 7M_1 + N - 1$

$= M + N - 1$

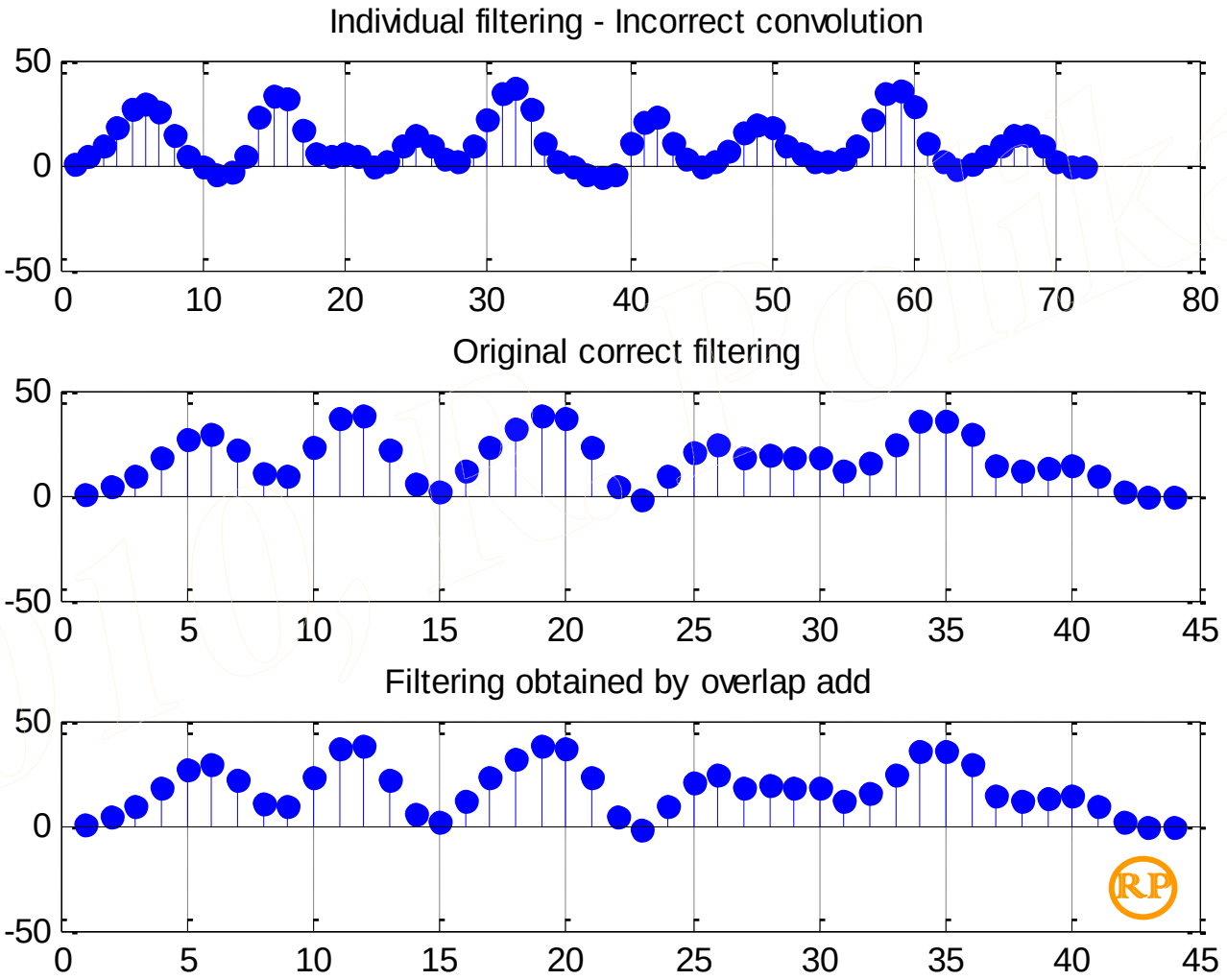
Example (Cont.)

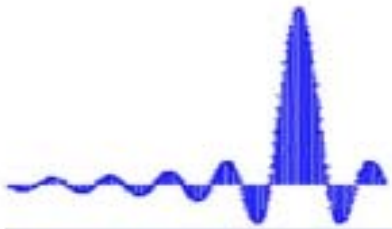
- So, here are the individual segments ordered in an overlap-add format

```
M=length(x); %Length of the original signal
M1=length(x1); % Length of each batch
N=length(h); %Length of the filter
K=length(y1); %Length of each filtered batch
L=M+N-1; %Length of the desired convolution
O=K-M1; % Amount of overlap

Y1=[zeros(1, 0*M1) y1 zeros(1, L-(1*M1)-O)];
Y2=[zeros(1, 1*M1) y2 zeros(1, L-(2*M1)-O)];
Y3=[zeros(1, 2*M1) y3 zeros(1, L-(3*M1)-O)];
Y4=[zeros(1, 3*M1) y4 zeros(1, L-(4*M1)-O)];
Y5=[zeros(1, 4*M1) y5 zeros(1, L-(5*M1)-O)];
Y6=[zeros(1, 5*M1) y6 zeros(1, L-(6*M1)-O)];
Y7=[zeros(1, 6*M1) y7 zeros(1, L-(7*M1)-O)];
Y8=[zeros(1, 7*M1) y8 zeros(1, L-(8*M1)-O)];

Y=Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8;
```



Example (Cont.)





Classification of sequences

Classification of finite length sequences:

conjugate symmetry relations

$$x_{cs}[n]=\frac{1}{2}(x[n]+x^{*}[<-n>_{N}]),\quad 0\le n\le N-1$$

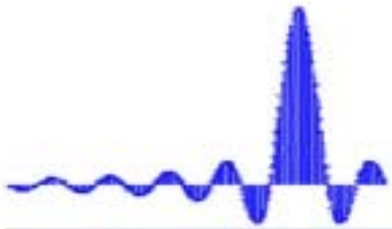
$$x_{ca}[n]=\frac{1}{2}(x[n]-x^{*}[<-n>_{N}]),\quad 0\le n\le N-1$$

$$\Big\}\quad x[n]=\frac{1}{2}(x_{cs}[n]+x_{ca}[n])$$

$$X_{cs}[k]=\frac{1}{2}(X[k]+X^{*}[<-k>_{N}]),\quad 0\le k\le N-1$$

$$X_{ca}[k]=\frac{1}{2}(X[k]-X^{*}[<-k>_{N}]),\quad 0\le k\le N-1$$

$$\Big\}\quad X[k]=\frac{1}{2}(X_{cs}[k]+X_{ca}[k])$$



Classification of sequences

other symmetry relations

$$x[n] = x[N-1-n] \qquad \text{symmetric}$$

$$x[n] = -x[N-1-n] \qquad \text{antisymmetric}$$

four types (see Mitra section 5.5.2):

- type 1: symmetric with even length
- type 2: symmetric with odd length
- type 3: asymmetric with even length
- type 4: asymmetric with odd length

Classification of sequences

Examples of sequences with the four types of geometric symmetry:

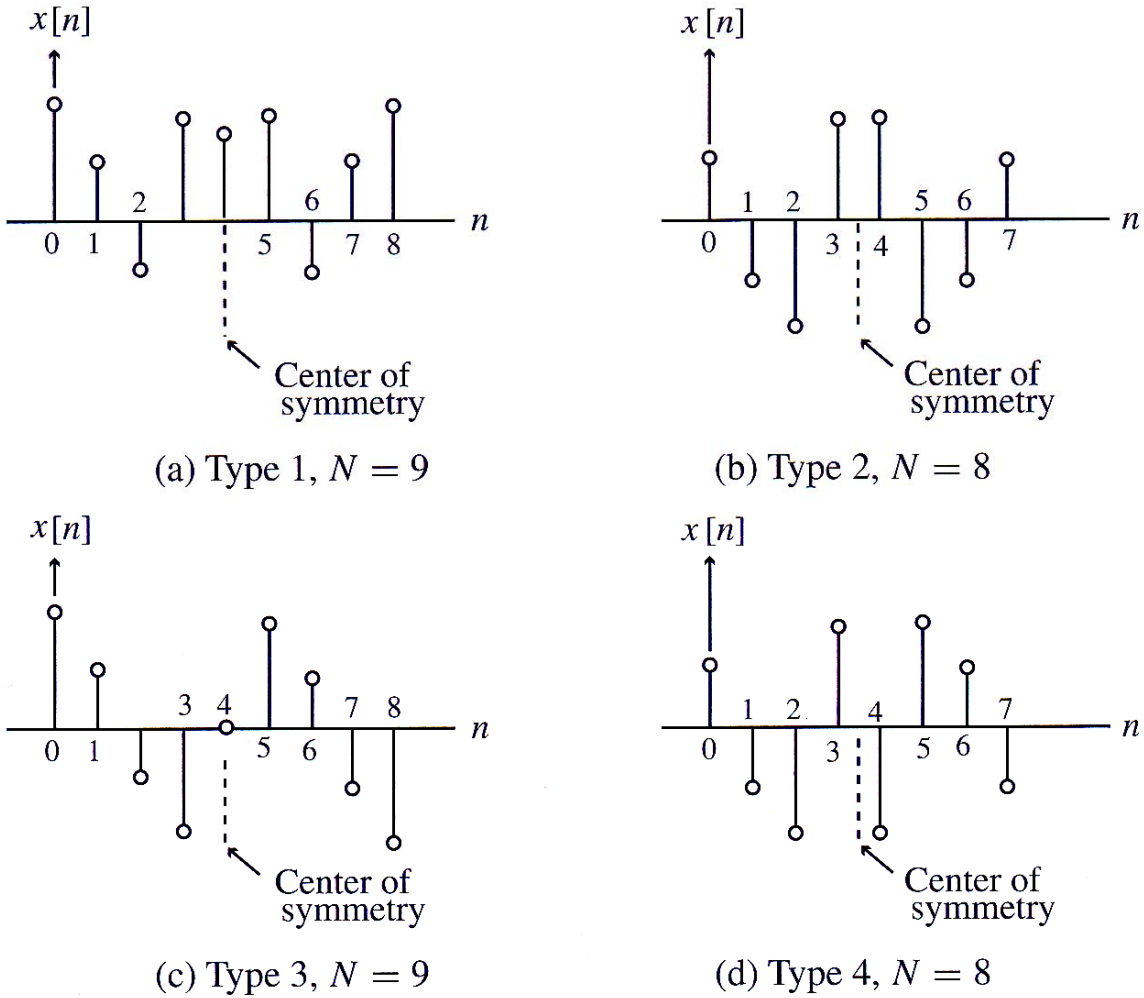
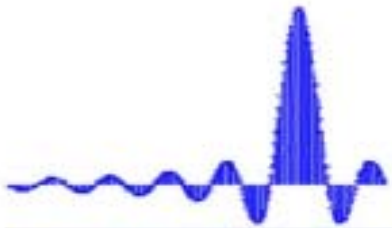


Figure 5.10: Illustration of the four types of geometric symmetry of a sequence.



Classification of sequences

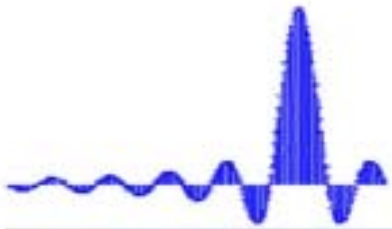
the DFT of a real (anti)symmetric sequence is the product of a phase term and an amplitude function (Mitra 5.5.2):

Type 1
$$X[k] = e^{-j(N-1)\pi k/N}\left\{ x\left[\frac{N-1}{2}\right] + 2\sum_{n=1}^{(N-1)/2} x\left[\frac{N-1}{2}-n\right]\cos\left(\frac{2\pi kn}{N}\right)\right\}$$

Type 2
$$X[k] = e^{-j(N-1)\pi k/N}\left\{ 2\sum_{n=1}^{N/2} x\left[\frac{N}{2}-n\right]\cos\left(\frac{\pi k(2n-1)}{N}\right)\right\}$$

Type 3
$$X[k] = e^{-j(N-1)\pi k/N}\left\{ 2\sum_{n=1}^{(N-1)/2} x\left[\frac{N-1}{2}-n\right]\sin\left(\frac{2\pi kn}{N}\right)\right\}$$

Type 4
$$X[k] = e^{-j(N-1)\pi k/N}\left\{ 2\sum_{n=1}^{N/2} x\left[\frac{N}{2}-n\right]\sin\left(\frac{\pi k(2n-1)}{N}\right)\right\}$$



Cosine transform

In the DFT of the different types the ***phase factor*** describes the ***length*** of the sequence, the ***amplitude function*** describes the ***time domain characteristics***. Extract the information via ***discrete cosine transforms***.

For type 2:

$$X_{DCT}[k] = \sum_{n=0}^{N-1} 2x[n]\cos\left(\frac{\pi k(2n+1)}{2N}\right), \qquad 0 \le k \le N-1$$

$$x_{DCT}[n] = \frac{1}{N}\sum_{k=0}^{N-1} \alpha[k]\, X_{DCT}[k]\cos\left(\frac{\pi k(2n+1)}{2N}\right), \qquad 0 \le n \le N-1$$

where: $\alpha[k] = \begin{cases} 1/2, & k = 0 \\ 1, & 1 \le k \le N-1 \end{cases}$

The even symmetrical DCT is used for data compression, in particular for images and video: JPEG, MPEG, H.261