EE3S1 Signal Processing – DSP

Lecture 7: Fast Fourier Transform (FFT) (Ch. 11)

Alle-Jan van der Veen

–

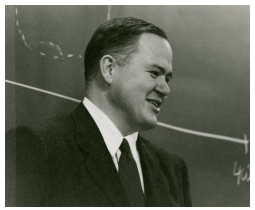
25 October 2025

Contents

- Matrix representation of the DFT [Ch. 10.4]
- Radix-2 FFT [Ch. 11.1]
- Fast convolution using the FFT [Ch. 2.7, 11.8]

Carl Gauss

James Cooley



John Tukey

Matrix representation of the DFT

Recall the DFT:

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

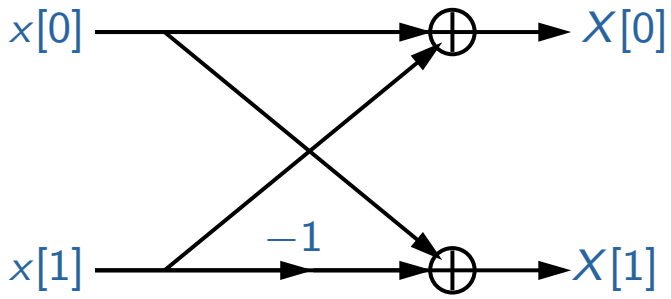
where $W_N = e^{-j\frac{2\pi}{N}}$. Due to its linearity, this expression can be written as a matrix-vector product:

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^{N-1} \\ W^0 & W^2 & W^4 & \cdots & W^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W^0 & W^{N-1} & W^{2(N-1)} & \cdots & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$

$$\Leftrightarrow \quad \boldsymbol{X} = \boldsymbol{F}_N\, \boldsymbol{x}$$

Example

$$\boldsymbol{F}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \Leftrightarrow$$



$$\boldsymbol{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

- Calculation of an $N \times N$ matrix-vector product generally requires N^2 complex operations (multiplications + additions)

  Suppose $N = 1000$, then $N^2 = 1$ million: very large!
- Calculating the DFT for $N = 2$ and $N = 4$ doesn't require any multiplications, just additions.

Inverse DFT

The IDFT is very similar:

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

In matrix form:

$$\begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix} = \frac{1}{N} \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^{-1} & W^{-2} & \cdots & W^{-(N-1)} \\ W^0 & W^{-2} & W^{-4} & \cdots & W^{-2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W^0 & W^{-(N-1)} & W^{-2(N-1)} & \cdots & W^{-(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{bmatrix}$$

$$\Leftrightarrow \quad \boldsymbol{x} = \boldsymbol{F}_N^{-1}\boldsymbol{X}$$

- Note that $\boldsymbol{F}_N^{-1} = \frac{1}{N}\boldsymbol{F}_N^H$, where ^{H} denotes the complex conjugate (=Hermitian) transpose.

Orthogonality

$$\boldsymbol{F}_N^{-1} = \frac{1}{N}\boldsymbol{F}_N^H \quad \Rightarrow \quad \boldsymbol{F}^H\boldsymbol{F} = N\,\boldsymbol{I}$$

- The columns of $\boldsymbol{F}$ are orthogonal to each other.
- $\boldsymbol{F}$ is a *unitary* matrix (except for the factor N)

Proof: the i,jth element of the product $\boldsymbol{F}^H\boldsymbol{F}$ is

$$(\boldsymbol{F}^H\boldsymbol{F})_{i,j} = \sum_{n=0}^{N-1} W_N^{-in} W_N^{jn} = \sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}(i-j)} = \begin{cases} N & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

$$\begin{bmatrix} 1 & 1 & 1 & \cdots 1 \\ \vdots & & \vdots & \vdots \\ 1 & W^{-i} & W^{-2i} & \cdots W^{-(N-1)i} \\ \vdots & & \vdots & \ddots \vdots \\ 1 & W^{-(N-1)} & W^{-2(N-1)} & \cdots W^{-(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} 1 \cdots 1 & \cdots 1 \\ 1 & W^{j} & \cdots W^{N-1} \\ 1 \cdots W^{2j} & \cdots W^{2(N-1)} \\ \vdots & \vdots & \ddots \vdots \\ 1 \cdots W^{(N-1)j} & \cdots W^{(N-1)(N-1)} \end{bmatrix} = N\boldsymbol{I}$$

Cyclic convolution in matrix form

$$y[n] = h[n] \circledast x[n] = \sum_{k=0}^{N-1} x[k]h[n-k \mod N]$$

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

$$\Leftrightarrow \quad \boldsymbol{y} = \mathcal{H}\,\boldsymbol{x}$$

Matrix $\mathcal{H}$ has a cyclic *Toeplitz* structure: constant along diagonals.

$\boldsymbol{F}$ diagonalizes a cyclic Toeplitz matrix

$$y[n] = h[n] \circledast x[n] \quad \Leftrightarrow \quad \boldsymbol{y} = \mathcal{H}\boldsymbol{x}$$

(Cyclic) convolution becomes product in frequency domain:
$Y[k] = H[k]X[k]$

$$\Leftrightarrow \begin{bmatrix} Y[0] \\ Y[1] \\ Y[2] \\ \vdots \\ Y[N-1] \end{bmatrix} = \underbrace{\begin{bmatrix} H[0] & & & & \\ & H[1] & & & \\ & & H[2] & & \\ & & & \ddots & \\ & & & & H[N-1] \end{bmatrix}}_{\boldsymbol{\Lambda}} \begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{bmatrix}$$

Use $\boldsymbol{Y} = \boldsymbol{F}\boldsymbol{y}$ and $\boldsymbol{X} = \boldsymbol{F}\boldsymbol{X}$. This demonstrates that $\boldsymbol{\Lambda} = \boldsymbol{F}^{-1}\mathcal{H}\boldsymbol{F}$ is diagonal

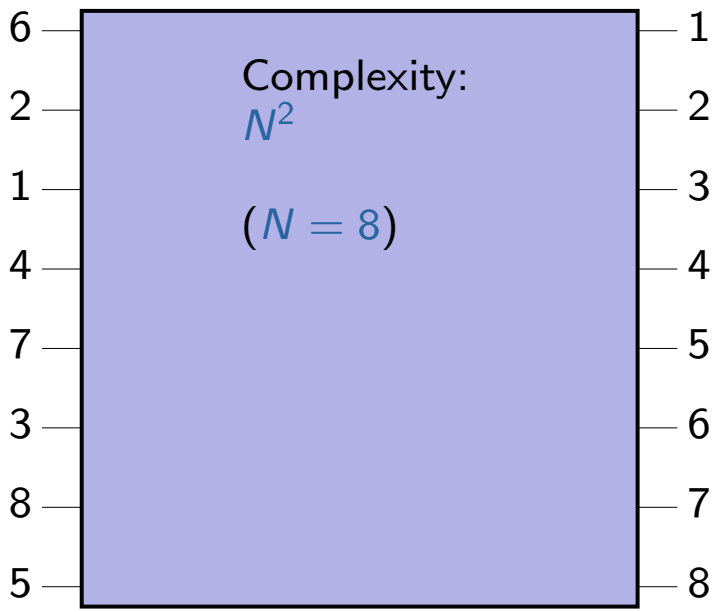
$\Rightarrow$ The columns of $\boldsymbol{F}$ are the eigenvectors of any cyclic Toeplitz matrix, $\boldsymbol{\Lambda}$ contains the eigenvalues.

Towards the FFT: Divide-and-conquer

Suppose we have a vector of N elements, and a task that requires $O(N^2)$ operations to complete.

- Example: sorting N numbers

  A simple algorithm requires N^2 comparisons



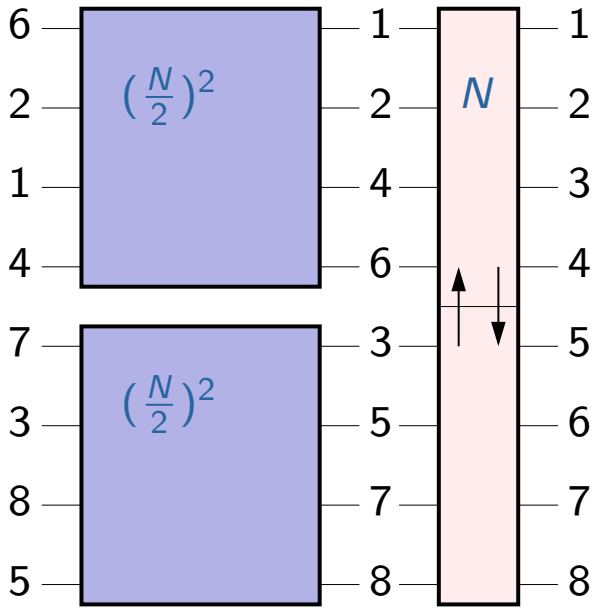
function sort(list)
for i $\leftarrow$ 1 **to** N
  find minimum in list
  sorted[i] $\leftarrow$ minimum
  remove minimum from list
end
return sorted

Divide-and-conquer (2)

- Suppose we can split the task into two similar tasks, each on $N/2$ elements, and a way to merge the results. (This is the basis of recursion.) Suppose the "merge" complexity is $O(N)$.

  Complexity: order $\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N$ [roughly half]

MergeSort algorithm




```
function sort(list)
   split list in [list1, list2]
   sorted1 ← sort(list1)
   sorted2 ← sort(list2)
   sorted ← merge(sorted1,sorted2)
   return sorted
end
```

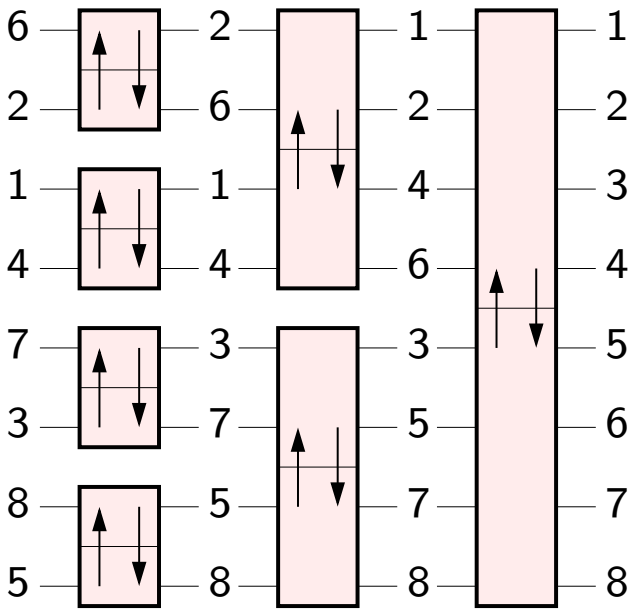
Divide-and-conquer (3)

MergeSort algorithm

- The recursion stops when we have $\frac{1}{2}N$ boxes, each comparing just 2 elements

  Complexity: $\frac{1}{2}N$. We need $\log_2(N)$ stages.

- Total complexity: order $N \log_2(N)$; for large N much smaller than N^2
- This worked because the "merge" step had complexity $O(N)$



Complexity: here $3N$
where $3 = \log_2(8)$ stages

Computing the DFT using Divide-and-Conquer

Application of Divide-and-Conquer to the DFT:

- Suppose we compute the DFT of two vectors with $\frac{1}{2}N$ elements, can we efficiently "merge" these results?

For ease of discussion, we assume that $N = 2^M$ is a power of two (otherwise, do zero padding on $x[n]$)

- Break down the N-point DFT to a cascade of smaller-size DFTs
- Compute the smaller-size DFT recursively
- Merge the smaller size DFT to get the N-point DFT

Computing the DFT using Divide-and-Conquer

- DFT is given by

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

- Split the summation into even and odd-indexed samples:

$$X[k] = \sum_{m=0}^{N/2-1} \underbrace{x[2m]}_{x_e[m]} e^{-j\frac{2\pi}{N}k(2m)} + \sum_{m=0}^{N/2-1} \underbrace{x[2m+1]}_{x_o[m]} e^{-j\frac{2\pi}{N}k(2m+1)}$$

- Rewrite the summation as follows:

$$X[k] = \underbrace{\sum_{m=0}^{N/2-1} x_e[m] e^{-j\frac{2\pi}{N/2}km}}_{X_e[k]} + e^{-j\frac{2\pi}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_o[m] e^{-j\frac{2\pi}{N/2}km}}_{X_o[k]}$$

Observation 1

$$X[k] = \underbrace{\sum_{m=0}^{N/2-1} x_e[m] e^{-j\frac{2\pi}{N/2}km}}_{X_e[k]} + e^{-j\frac{2\pi}{N}k} \underbrace{\sum_{m=0}^{N/2-1} x_o[m] e^{-j\frac{2\pi}{N/2}km}}_{X_o[k]}$$

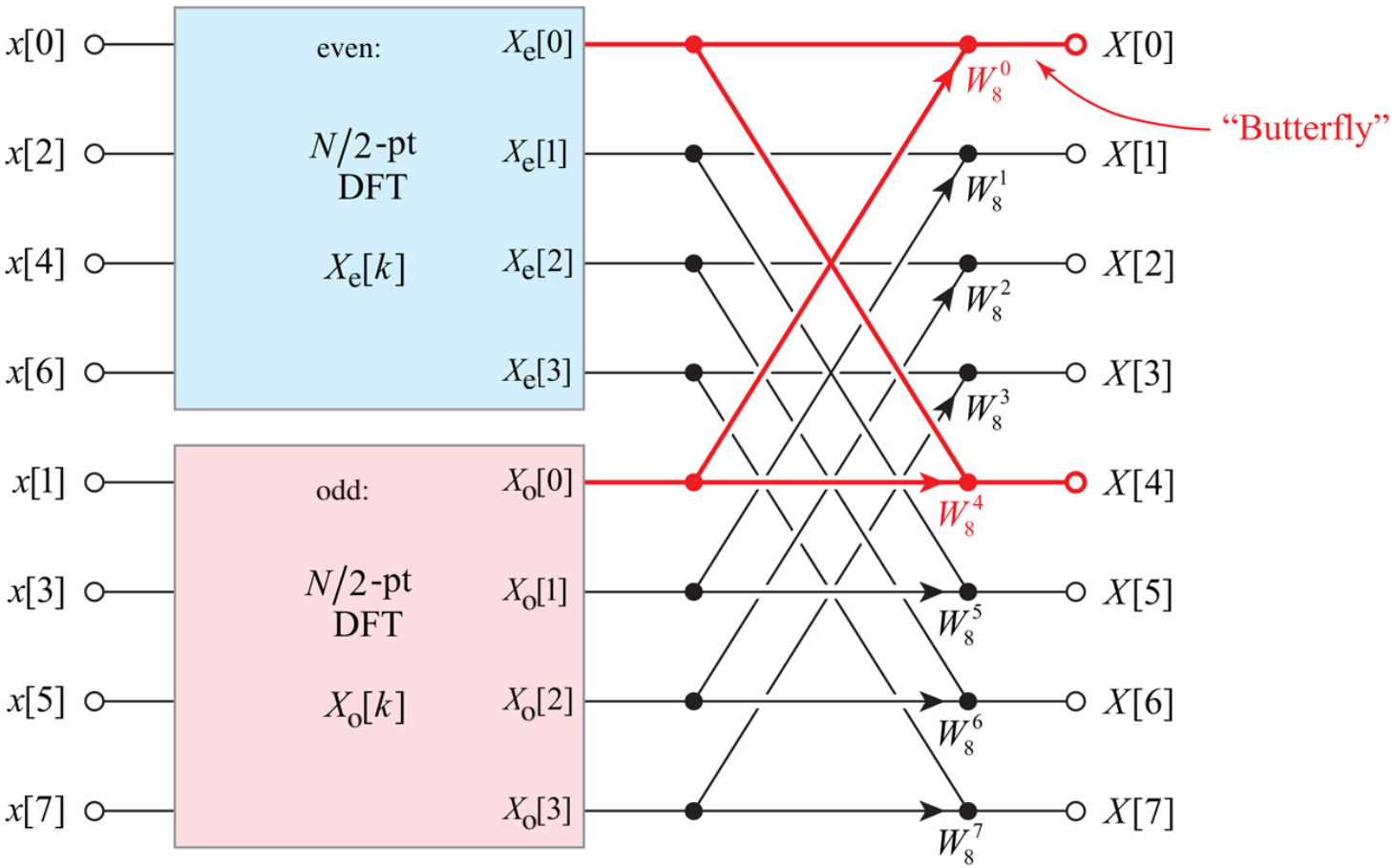
We compute the DFT of $x_e[m]$ and $x_0[m]$, each of $\frac{1}{2}N$ samples, then combine.

- The DFT of $x_e[m]$ is $X_e[k]$ with $\frac{1}{2}N$ entries ($k = 0, \cdots, \frac{1}{2}N - 1$)
  For $k \geq \frac{1}{2}N$, don't recompute but use periodicity of the DFT:

  $$X_e[k] = X_e[k - \tfrac{1}{2}N] \qquad (k = \tfrac{1}{2}N, \cdots, N-1)$$

  So we compute $X_e[k]$ only for $k < \frac{1}{2}N$, then duplicate.
  (Same for $X_o[k]$)
- The $e^{-j\frac{2\pi}{N}k}$ are *twiddle factors* [caused by delay of 1 sample]

Butterfly diagram: $N = 8$



Observation 2

Regarding the twiddle factors $W_N^k = e^{-j\frac{2\pi}{N}k}$:

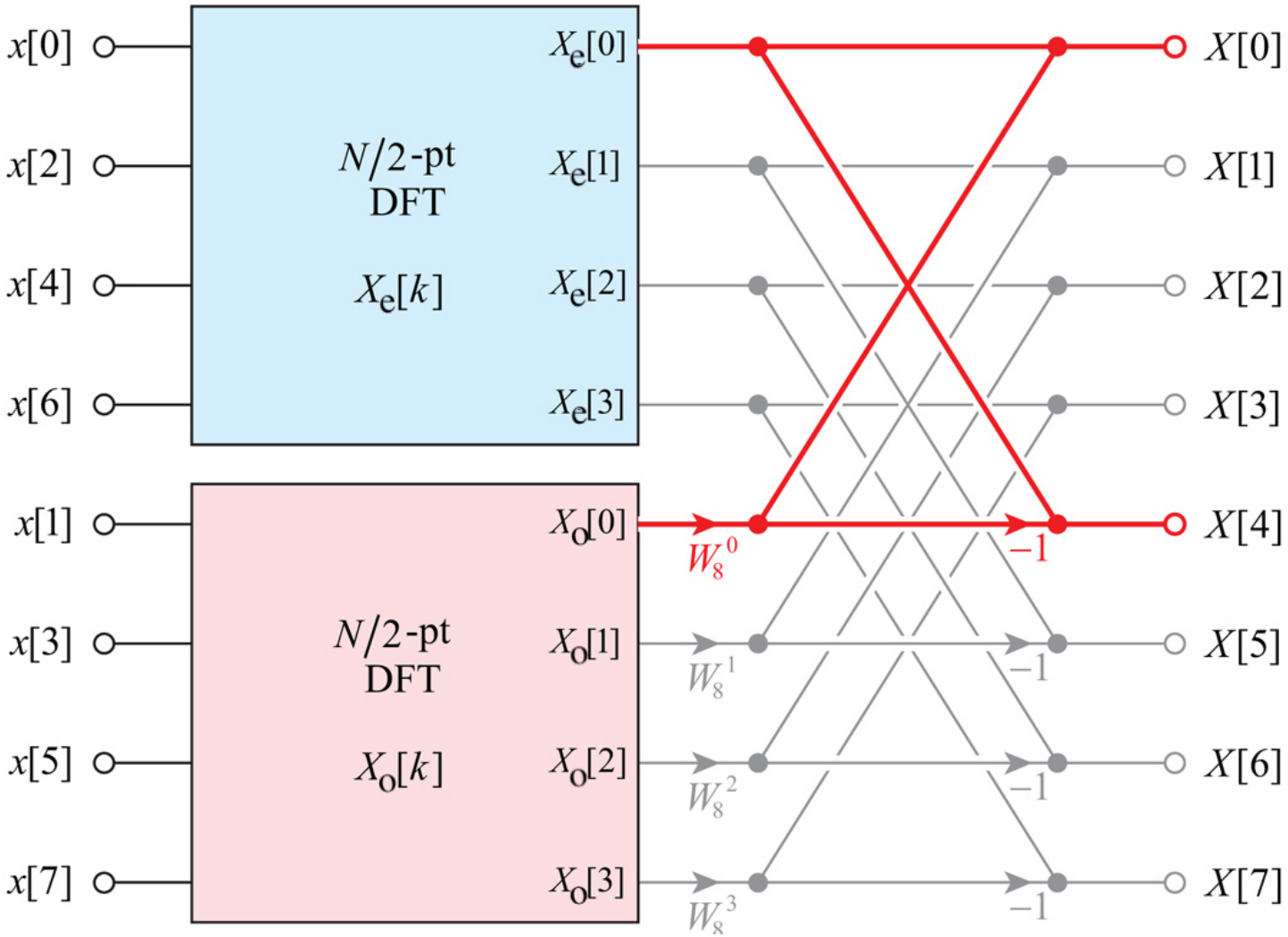
$$\text{for } k \geq N/2: \quad e^{-j\frac{2\pi}{N}k} = -e^{-j\left(\frac{2\pi}{N}k - \pi\right)} = -e^{-j\frac{2\pi}{N}(k-\frac{1}{2}N)}$$

or

$$k \geq N/2: \quad W_N^k = -W_N^{k-\frac{1}{2}N}$$

So we only need to compute the twiggle factors for $k = 0, \cdots, \frac{1}{2}N - 1$. (These are usually precomputed and stored.)

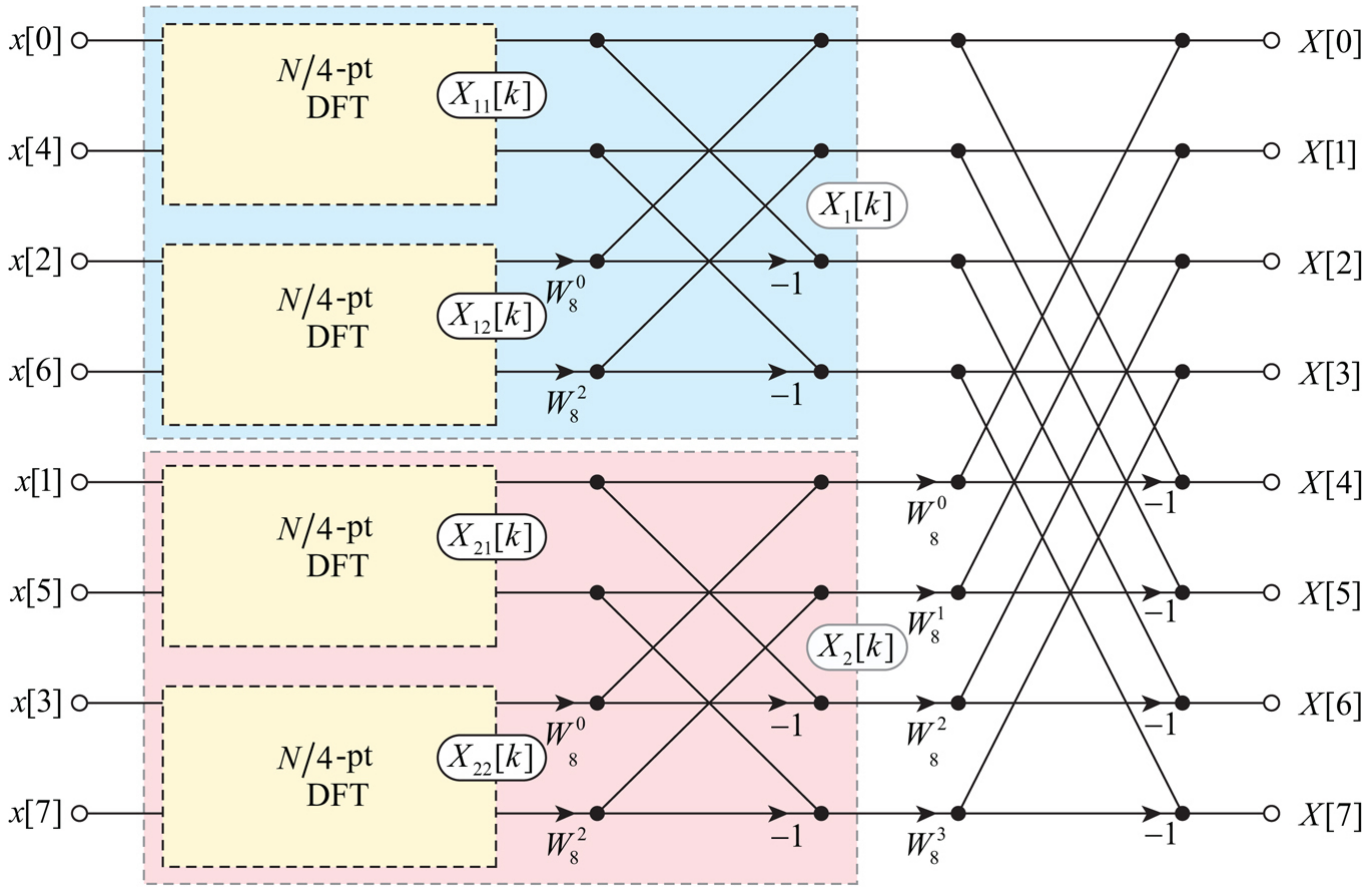
Butterfly diagram: $N = 8$



The butterflies are actually DFT($N = 2$) operations! (see slide **??**)

Entering a recursion

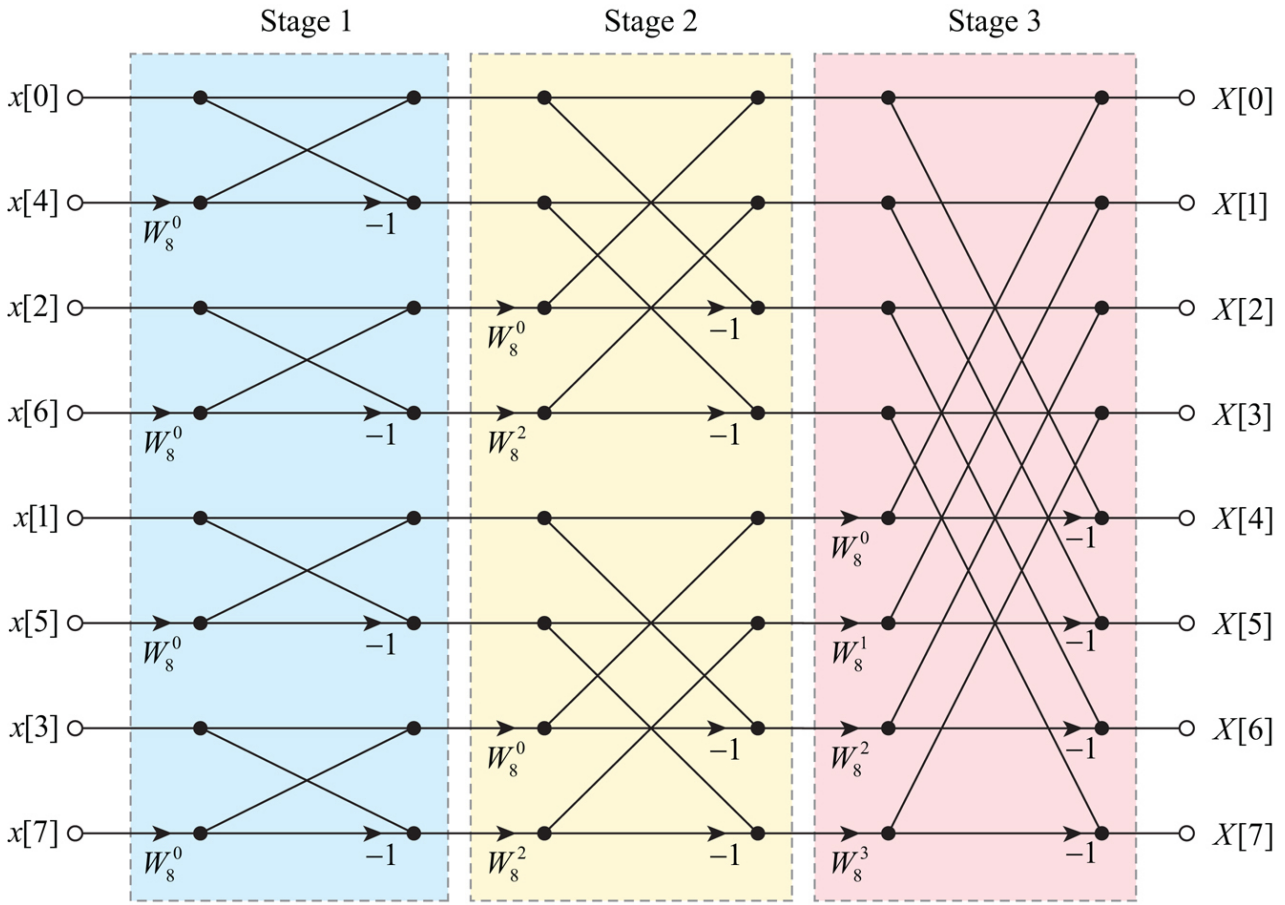
Apply the same idea to the DFT of order $N/2$!



Twiggle factors: used that $W_4^1 = W_8^2$: need only 1 table for this.

Radix-2 FFT

Continue the recursion until we get to size $N = 2$



This is the Radix-2 FFT, via *decimation in time* (meaning downsampling). Number of stages: $r = \log_2(N)$

Radix-2 FFT Implementation

- Ordering of the samples follows from the repeated even/odd partitioning (start at the right).

  It can be obtained via *bit-reversal*:

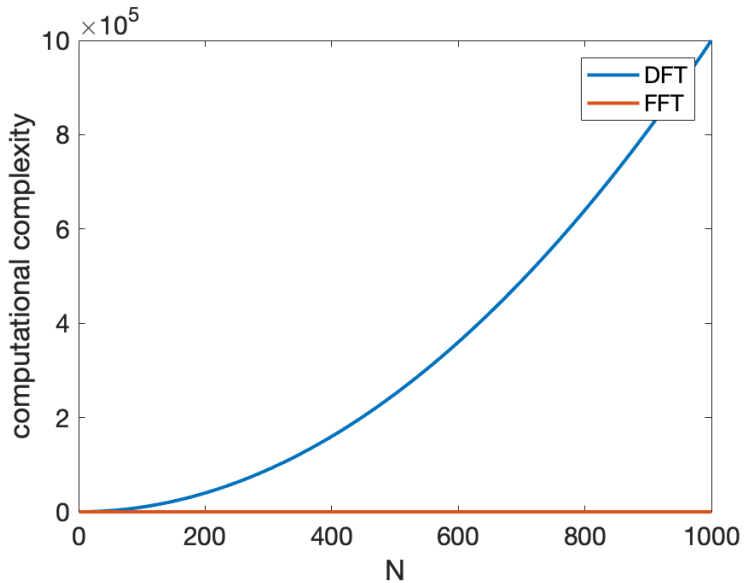
$$
\begin{array}{lllll}
0 = [0\ 0\ 0] & \rightarrow & [0\ 0\ 0] = 0 & \rightarrow & x[0] \\
1 = [0\ 0\ 1] & \rightarrow & [1\ 0\ 0] = 4 & \rightarrow & x[4] \\
2 = [0\ 1\ 0] & \rightarrow & [0\ 1\ 0] = 2 & \rightarrow & x[2] \\
3 = [0\ 1\ 1] & \rightarrow & [1\ 1\ 0] = 6 & \rightarrow & x[6] \\
4 = [1\ 0\ 0] & \rightarrow & [0\ 0\ 1] = 1 & \rightarrow & x[1] \\
5 = [1\ 0\ 1] & \rightarrow & [1\ 0\ 1] = 5 & \rightarrow & x[5] \\
6 = [1\ 1\ 0] & \rightarrow & [0\ 1\ 1] = 3 & \rightarrow & x[3] \\
7 = [1\ 1\ 1] & \rightarrow & [1\ 1\ 1] = 7 & \rightarrow & x[7]
\end{array}
$$

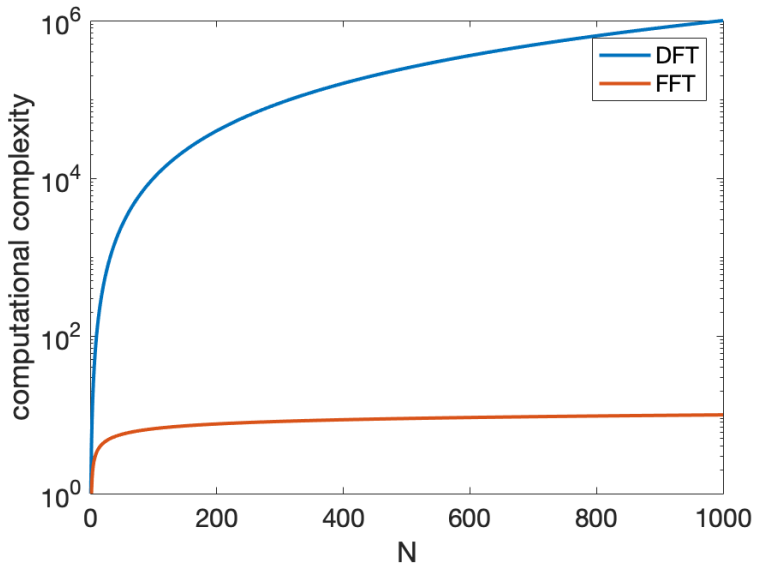
- Use zero padding to make $N = 2^r$ for some r, leading to r stages

Computational complexity of Radix-2 FFT

Computational complexity:

- We have $r = \log_2 N$ stages, each with $\frac{1}{2}N$ butterflies, requiring $\frac{1}{2}N$ complex multiplications and $\frac{3}{2}N$ complex additions
- Total complexity is $\frac{1}{2}N \log_2 N$ complex multiplications and $\frac{3}{2}N \log_2 N$ complex additions, so overall $O(N \log_2 N)$
- For large N, this is much smaller than N^2





Decimation-in-Frequency FFT

Alternative to splitting $x[n]$ into even/odd, split into lower/higher blocks of $\frac{1}{2}N$ samples:

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \\
&= \sum_{n=0}^{N/2-1} x[n] e^{-j\frac{2\pi}{N}kn} + \sum_{n=0}^{N/2-1} x[n+N/2] e^{-j\frac{2\pi}{N}k(n+N/2)} \\
&= \sum_{n=0}^{N/2-1} x[n] e^{-j\frac{2\pi}{N}kn} + e^{-j\frac{2\pi}{N}kN/2} \sum_{n=0}^{N/2-1} x[n+\frac{1}{2}N] e^{-j\frac{2\pi}{N}kn} \\
&= \sum_{n=0}^{N/2-1} x[n] e^{-j\frac{2\pi}{N}kn} + (-1)^k \sum_{n=0}^{N/2-1} x[n+\frac{1}{2}N] e^{-j\frac{2\pi}{N}kn}
\end{aligned}
$$

Decimation-in-Frequency FFT

Split into even and odd k resulting in two DFT's of size $\frac{1}{2}N$:

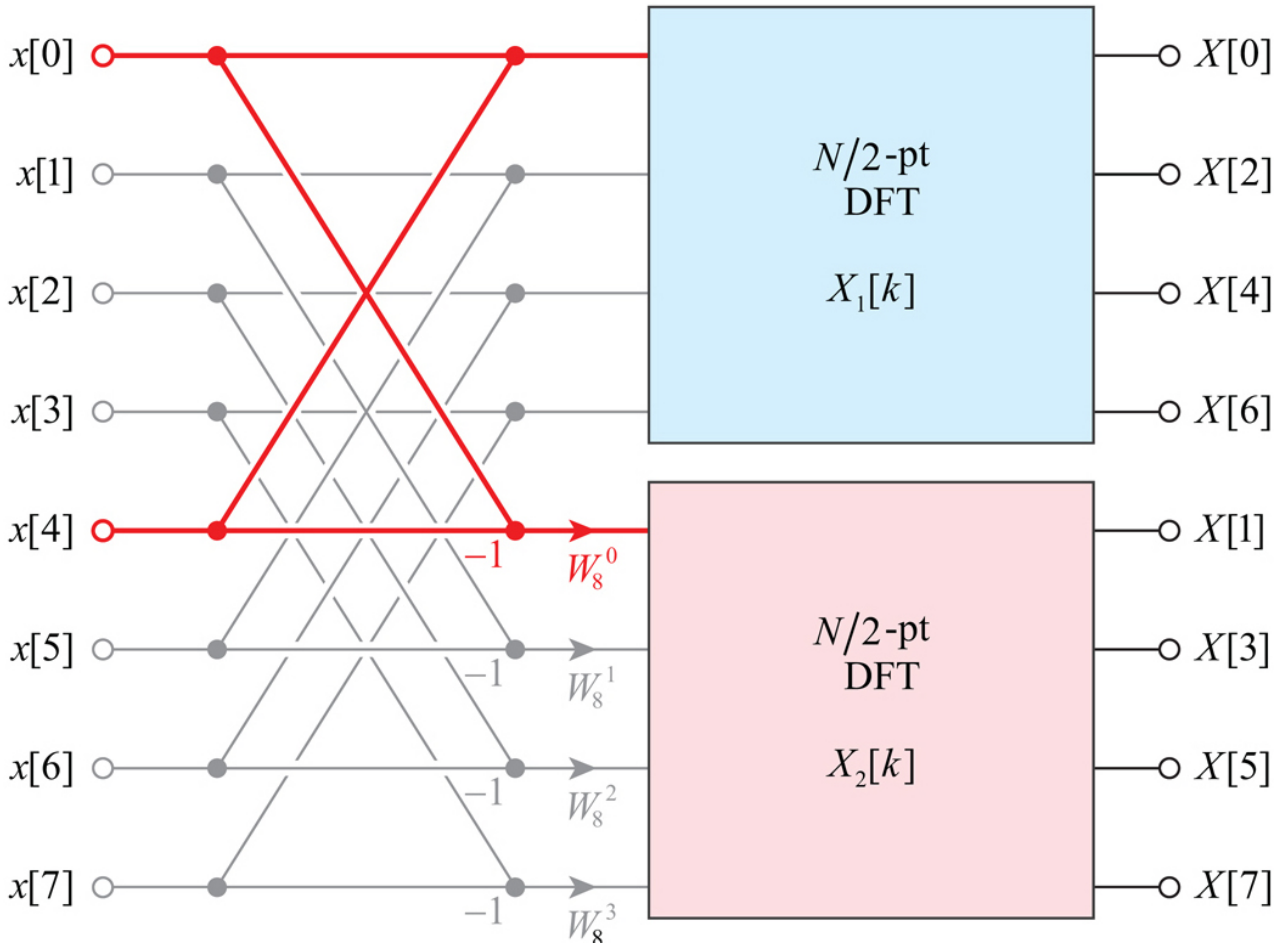
- $k = 2p$, for $0 \leq p < N/2$:

$$X[2p] = \sum_{n=0}^{N/2-1} x[n] e^{-j\frac{2\pi}{N}2pn} + \sum_{n=0}^{N/2-1} x[n+\frac{1}{2}N] e^{-j\frac{2\pi}{N}2pn}$$
$$= \sum_{n=0}^{N/2-1} (x[n] + x[n+\frac{1}{2}N]) e^{-j\frac{2\pi}{N/2}pn}$$

- $k = 2p+1$, for $0 \leq p < N/2$:

$$X[2p+1] = \sum_{n=0}^{N/2-1} x[n] e^{-j\frac{2\pi}{N}(2p+1)n} - \sum_{n=0}^{N/2-1} x[n+\frac{1}{2}N] e^{-j\frac{2\pi}{N}(2p+1)n}$$
$$= \sum_{n=0}^{N/2-1} [(x[n] - x[n+\frac{1}{2}N]) e^{-j\frac{2\pi}{N}n}] e^{-j\frac{2\pi}{N/2}pn}$$

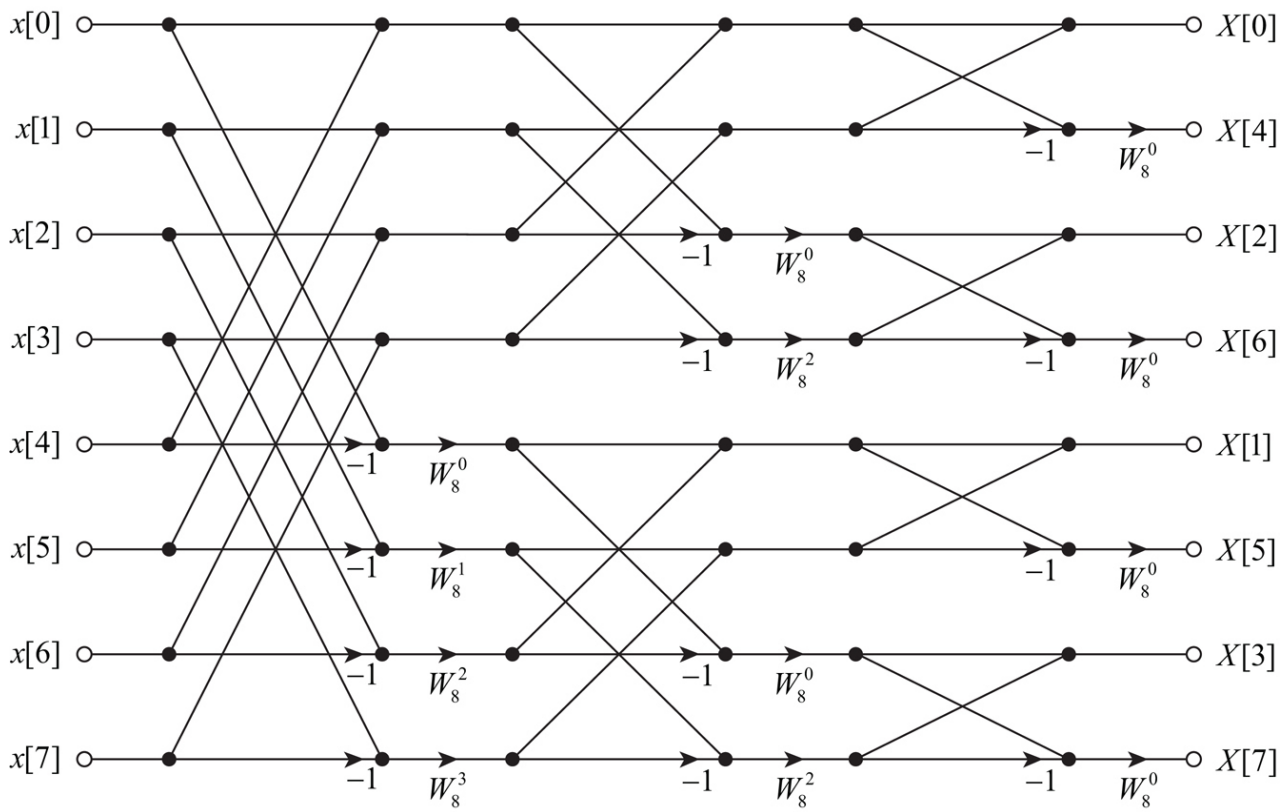
Decimation-in-Frequency FFT ($N = 8$)



This is a "transposed" network of the Decimation-in-Time structure

Radix-2 Decimation-in-Frequency FFT ($N = 8$)

Complete the recursion on the $\frac{1}{2}N$ blocks:



Cooley-Tukey Decomposition (mixed-radix FFT)

Beyond radix-2: Suppose we can factor $N = PQ$. We can then reduce complexity by grouping $x[n]$ into P segments of length Q:

Let $n = Pq + p$ and $k = Qs + r$, then

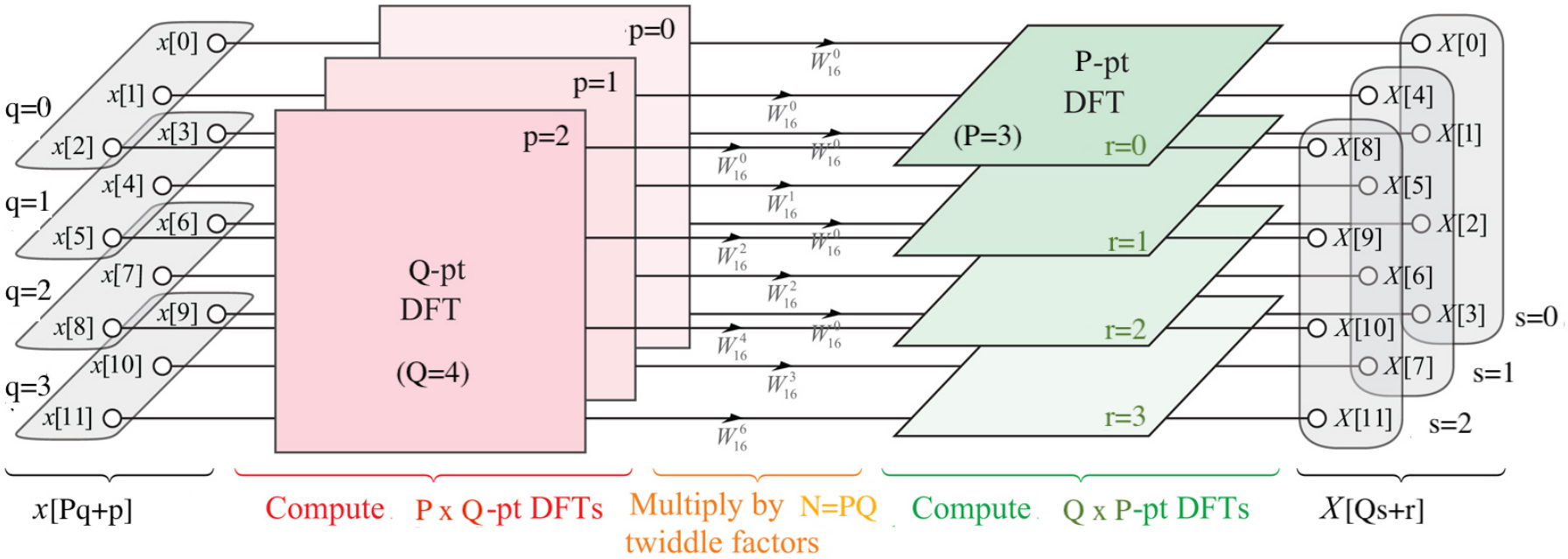
$$e^{-j\frac{2\pi}{N}kn} = e^{-j\frac{2\pi}{N}(Qs+r)(Pq+p)} = e^{-j\frac{2\pi}{N}(\cancel{Nsq}+rp+Qsp+Prq)} = e^{-j\frac{2\pi}{N}rp} e^{-j\frac{2\pi}{Q}rq} e^{-j\frac{2\pi}{P}sp}$$

$$\begin{aligned} X[k] = X[Qs+r] &= \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} x[Pq+p] e^{-j\frac{2\pi}{N}rp} e^{-j\frac{2\pi}{Q}rq} e^{-j\frac{2\pi}{P}sp} \\ &= \sum_{p=0}^{P-1} e^{-j\frac{2\pi}{N}rp} \left(\sum_{q=0}^{Q-1} \underbrace{x[Pq+p]}_{x_p[q]} e^{-j\frac{2\pi}{Q}rq} \right) e^{-j\frac{2\pi}{P}sp} \\ &= \sum_{p=0}^{P-1} \left(e^{-j\frac{2\pi}{N}rp} X_p[r] \right) e^{-j\frac{2\pi}{P}sp} \quad = Y_r[s] \end{aligned}$$

First P DFTs of size Q, then twiggle, then Q DFTs of size P

Cooley-Tukey Decomposition (mixed-radix FFT)

$N = 12$



- Generalizes radix-2 for which $P = 2$ or $Q = 2$
- This can be used to develop FFTs of more arbitrary sizes (recursively to small prime factors).

Summary

- FFT is an algorithm for calculating the DFT and its inverse (using the same approach)
- FFT reduces the complexity from N^2 to $N \log_2 N$ by taking advantage of $O(N)$ "merge" properties
- Since DFT can be used to perform filtering and spectral analysis, FFT helps to implement these important operations efficiently

This has enabled the digital revolution with many applications that otherwise would not exist (jpg, mp3, wifi, radar, MRI, $\cdots$)

Convolution of long sequences

Overlap-add method [ch. 2.7]

In some cases, $x[n]$ is long and we process the convolution in blocks (segments) $x_i[n]$ of the input data, each of length L.

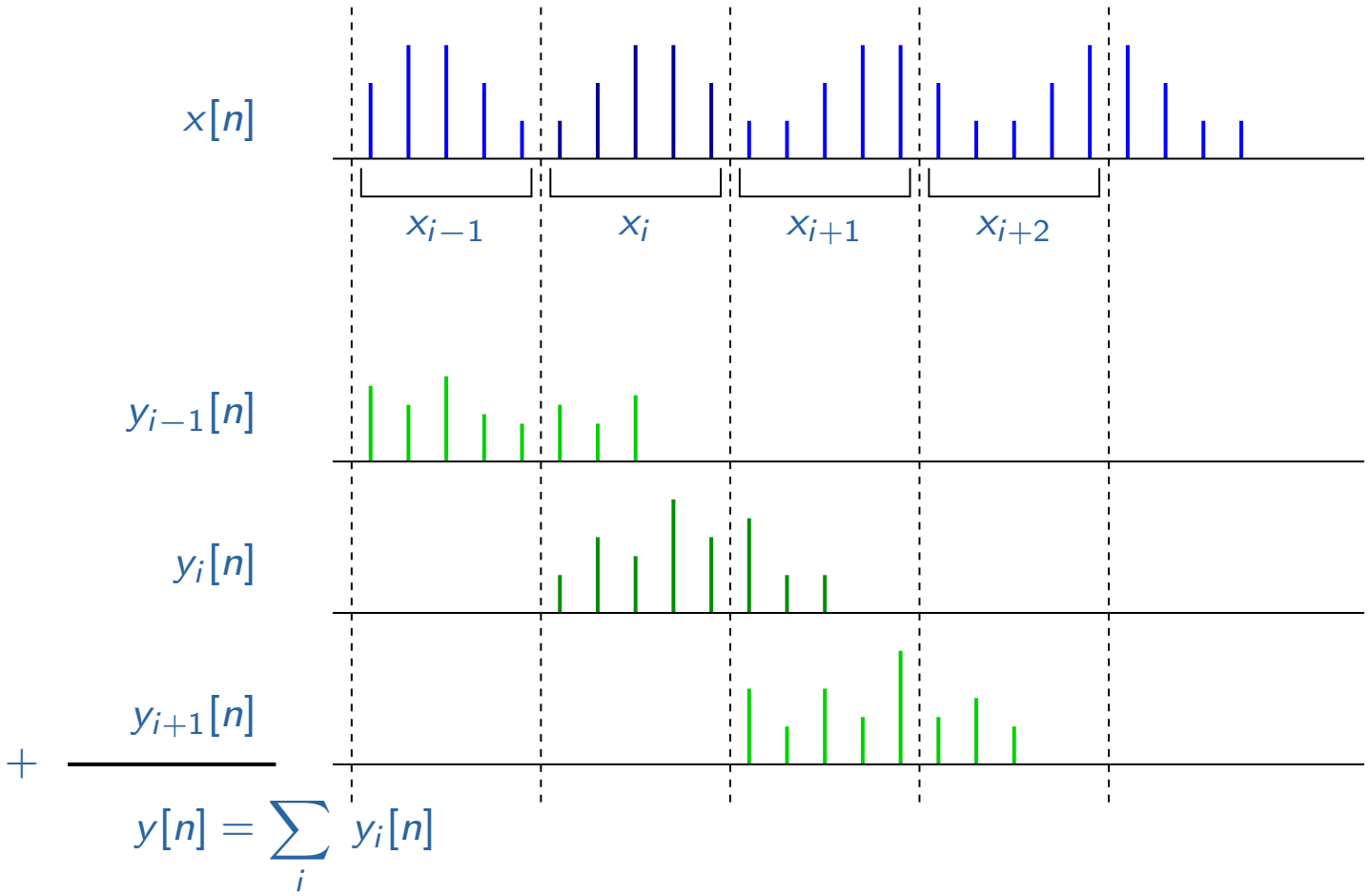
$$x[n] = \sum_i x_i[n] \quad \Rightarrow \quad y[n] = \sum_i x_i[n] * h[n]$$

If $h[n]$ has length M, then $y_i[n] = x_i[n] * h[n]$ has length $L + M - 1$

$\Rightarrow$ a "tail" of $M - 1$ samples of $y_i[n]$ will extend into the next segment.

Convolution of long sequences

Overlap-add method



Using zero padding, $y_i[n] = x_i[n] * h[n]$ can be implemented using a cyclic convolution, which can be efficiently calculated using the FFT.

Computational complexity

Let $h[n]$ be of length M.

Split $x[n]$ into $\lceil \frac{N}{L} \rceil$ segments of length $L \geq M$.

For each segment:

	Complexity:
Zero pad $x_i[n]$ and $h[n]$ to length $M + L - 1$	
FFT on the extended $x_i[n]$ and $h[n]$ (once)	$(M + L - 1) \log_2(M + L - 1)$
Multiply in frequency domain: $Y_i[k] = X_i[k]\, H[k]$	$M + L - 1$
Inverse FFT to obtain $y_i[n]$ of length $M + L - 1$	$(M + L - 1) \log_2(M + L - 1)$
Sum the results: $y[n] = \sum y_i[n]$	$M - 1$
	$\approx 2L \log_2 L + 5L$, if $M = L$

Total complexity: $O(N \log_2 L)$ rather than $O(NM)$ for a direct implementation.

To do:

- Study the covered parts of chapter 11
- Try to make exercise ...

Next lecture, we wrap up with a selected topic.