Lecture I213E – Class 11

# Discrete Signal Processing

Sakriani Sakti

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY
1990

# Course Materials

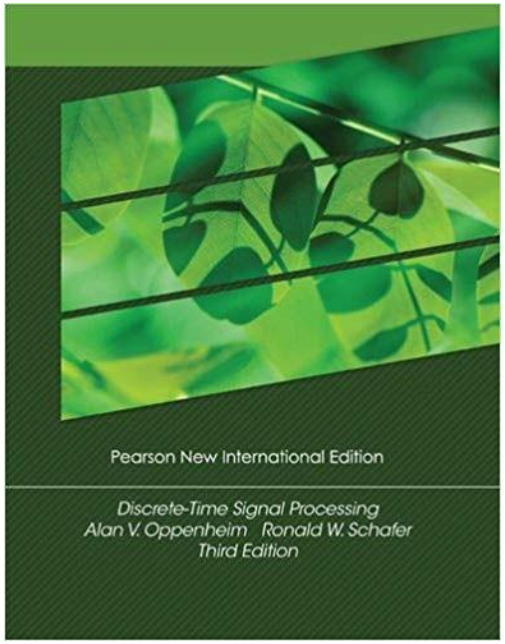
- **Materials**

  → Lecture notes will be uploaded before each lecture
  https://jstorage-2018.jaist.ac.jp/s/PGXRrC7iFmN2FWo
  Pass: dsp-i213e-2022
  (Slide Courtesy of Prof. Nak Young Chong)

- **References**

  → Chi-Tsong Chen:
  **Linear System Theory and Design,** 4th Ed.,
  Oxford University Press, 2013.

  → Alan V. Oppenheim and Ronald W. Schafer:
  **Discrete-Time Signal Processing,** 3rd Ed.,
  Pearson New International Ed., 2013.



# Related Courses & Prerequisite

- **Related Courses**
  - → I212 Analysis for Information Science
  - → I114 Fundamental Mathematics for Information Science

- **Prerequisite**
  - → None

# Evaluation

- **Viewpoint of evaluation**

  → Students are able to understand:

  - Basic principles in modeling and analysis of linear time-invariant systems
  - Applications of mathematical methods and tools to different signal processing problems.

- **Evaluation method**

  → Homework, term project, midterm exam, and final exam

- **Evaluation criteria**

  → Homework/labs (30%), term project (30%)
  midterm exam (15%), and final exam (25%)

# Contact

- **Lecturer**

  → Sakriani Sakti

- **TA**

  **Tutorial hours & Term project**

  → WANG Lijun (s2010026)

  → TANG Bowen (s2110411)

  **Homework**

  → PUTRI Fanda Yuliana (s2110425)

- **Contact Email**

  → dsp-i213e-2022@ml.jaist.ac.jp

# Schedule

- **December 8th, 2022 – February 9th, 2023**

- **Lecture Course Term 2-2**

  → Tuesday 9:00 — 10:40

  → Thursday 10:50 — 12:30

- **Tutorial Hours**

  → Tuesday 13:30-15:10

# Schedule

**Dec**

| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

**Jan**

| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | ✖ | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | ✖ | 25 | 26 | 27 | 28 |
| 29 | 30 | 31 | | | | |

**Feb**

| Sun | Mon | Tue | Wed | Thu | Fri | Sat |
|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| 19 | 20 | 21 | 22 | 23 | 24 | 25 |
| 26 | 27 | 28 | | | | |

Lecture:
Tuesday 9:00 — 10:40
Thursday 10:50 — 12:30

Additional Lecture:
Monday 17:10 — 18:50

Tutorial:
Tuesday 13:30 — 15:10

Course review &
term project evaluation
(on tutorial hours)

Midterm & final exam
Thursday 10:50 — 12:30

# Syllabus

| Class | Date | Lecture Course<br>Tue 9:00 — 10:40 / Thr 10:50 — 12:30 | Tutorial Hours<br>Tue 13:30 — 15:10 |
|---|---|---|---|
| 1 | 12/08 | Introduction to Linear Systems with Applications to Signal Processing | |
| 2 | 12/13 | State Space Description | ○ |
| 3 | 12/15 | Linear Algebra | |
| 4 | 12/20 | Quantitative Analysis (State Space Solutions) and Qualitative Analysis (Stability) | ○ |
| 5 | 12/22 | Discrete-time Signals and Systems | |
| X | 01/05 | | |
| 6 | 01/10 | Discrete-time Fourier Analysis | |
| 7 | 01/10* | Review of Discrete-time Linear Time-Invariant Signals and Systems (on Tutorial Hours) | |
| | 01/12 | Midterm Exam | |
| 8 | 01/17 | Sampling and Reconstruction of Analog Signals | ○ |
| 9 | 01/19 | z-Transform | |
| X | 01/24 | | ○ |
| 10 | 01/31 | Discrete Fourier Transform | ○ |
| 11 | 02/02 | FFT Algorithms | |
| 12 | 02/06 | Implementation of Digital Filters | |
| 13 | 02/07 | Digital Signal Processors and Design of Digital Filters | |
| 14 | 02/07* | Review of the Course and Term Project Evaluation (on Tutorial Hours) | |
| | 02/09 | Final exam | |

# Class 11
# Fast Fourier Transform

# Discrete Fourier Transform

# Discrete Fourier Transform

- **DFT**

DFT of an $N$-point sequence $x_n$, $n = 0, 1, 2, \ldots, N-1$ is defined as

$$X_k = \sum_{n=0}^{N-1} x_n \, e^{-j\frac{2\pi k}{N} n} \qquad k = 0, 1, 2, \cdots, N-1$$

- An $N$-point sequence yields an $N$-point transform
- $X_k$ can be expressed as an *inner product*:

$$X_k = \begin{bmatrix} 1 & e^{-j\frac{2\pi k}{N}} & e^{-j\frac{2\pi k}{N}2} & \ldots & e^{-j\frac{2\pi k}{N}(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

# Discrete Fourier Transform

- **DFT**

Notation: $W_N = e^{-j\frac{2\pi}{N}}$. Hence,

$$X_k = \begin{bmatrix} 1 & W_N^k & W_N^{2k} & \dots & W_N^{(N-1)k} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix}$$

➡ By varying $k$ from 0 to $N-1$ and combining the $N$ inner products, we get the following:

➡ **W** is an $N \times N$ matrix, called as the "DFT Matrix"

# DFT Matrix

- **Matrix W**

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & \cdots & W_N^{2(N-1)} \\ & & & \vdots & \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix}_{N \times N}$$

➡ The notation $\mathbf{W}_N$ is used if we want to make the size of the DFT matrix explicit

# Complexity of DFT

- **Multiplications**
  - ➡ Each inner product requires $N$ complex multiplications
    - There are $N$ inner products

    Hence we require $N^2$ multiplications
  - ➡ However, the first row and first column are all 1s, and *should not be counted as multiplications*
    - There are $2N-1$ such instances

    Hence, the number of complex multiplications is $N^2-2N+1$, i.e., $(N-1)^2$
- **Additions**
  - ➡ Each inner product requires $N-1$ complex additions
    - There are $N$ inner products

    Hence we require $N(N-1)$ complex additions

# Complexity of DFT

- **Total Operation**
  - No. of complex multiplications: $(N-1)^2$
    No. of complex additions: $N(N-1)$
  - For large $N$,

    $(N-1)^2 \approx N^2$

    $N(N-1) \approx N^2$

- **Complexity**
  - Hence both multiplications and additions are $O(N^2)$

    If $N = 10^3$, then $O(N^2) = 10^6$, i.e., a million!

    This makes the straightforward method slow and impractical even for a moderately long sequence

# Fast Fourier Transform

# Fast Fourier Transform

- **FFT**

  ➡ The numerical computation of the DFT for long sequence is prohibitively time consuming.

  ➡ Several algorithms have been developed to efficiently compute the DFT --- collectively called Fast Fourier Transform (FFT) algorithms.

  ➡ J.W. Cooley and J.W. Tukey, “An algorithm for the machine computation of complex Fourier series,” Mathematical Computations, 19:297-301, April 1965.

# Example

- **4-point DFT**

$$X(k)=\sum_{n=0}^{3}x(n)W_4^{nk},\quad 0\le k\le 3;\quad W_4=e^{-j2\pi/4}=-j$$

$$\begin{bmatrix}X(0)\\X(1)\\X(2)\\X(3)\end{bmatrix}=\begin{bmatrix}W_4^0 & W_4^0 & W_4^0 & W_4^0\\W_4^0 & W_4^1 & W_4^2 & W_4^3\\W_4^0 & W_4^2 & W_4^4 & W_4^6\\W_4^0 & W_4^3 & W_4^6 & W_4^9\end{bmatrix}\begin{bmatrix}x(0)\\x(1)\\x(2)\\x(3)\end{bmatrix}$$

*requires* 16 *complex multiplications*

➡ Using periodicity $W_4^0=W_4^4=1;\quad W_4^1=W_4^9=-j$

$$W_4^2=W_4^6=-1;\quad W_4^3=j$$

$$\begin{bmatrix}X(0)\\X(1)\\X(2)\\X(3)\end{bmatrix}=\begin{bmatrix}1 & 1 & 1 & 1\\1 & -j & -1 & j\\1 & -1 & 1 & -1\\1 & j & -1 & -j\end{bmatrix}\begin{bmatrix}x(0)\\x(1)\\x(2)\\x(3)\end{bmatrix}$$

# Example

- **4-point DFT**
  - ➡ Using symmetry

$$X(0) = x(0) + x(1) + x(2) + x(3) = \underbrace{[x(0) + x(2)]}_{g_1} + \underbrace{[x(1) + x(3)]}_{g_2}$$

$$X(1) = x(0) - jx(1) - x(2) + jx(3) = \underbrace{[x(0) - x(2)]}_{h_1} - j\underbrace{[x(1) - x(3)]}_{h_2}$$

$$X(2) = x(0) - x(1) + x(2) - x(3) = \underbrace{[x(0) + x(2)]}_{g_1} - \underbrace{[x(1) + x(3)]}_{g_2}$$

$$X(3) = x(0) + jx(1) - x(2) - jx(3) = \underbrace{[x(0) - x(2)]}_{h_1} + j\underbrace{[x(1) - x(3)]}_{h_2}$$

| *Step 1* | *Step 2* |
|---|---|
| $g_1 = x(0) + x(2)$ | $X(0) = g_1 + g_2$ |
| $g_2 = x(1) + x(3)$ | $X(1) = h_1 - jh_2$ |
| $h_1 = x(0) - x(2)$ | $X(2) = g_1 - g_2$ |
| $h_2 = x(1) - x(3)$ | $X(3) = h_1 + jh_2$ |

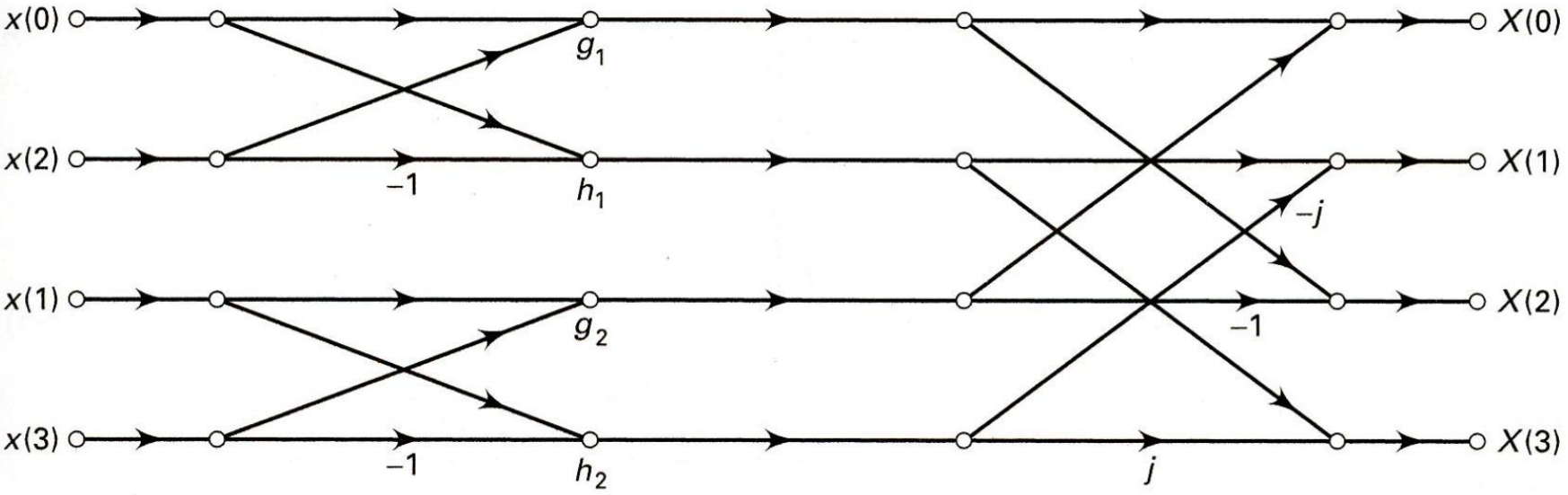
*requires only* 2 *complex multiplications*

# Example

- **4-point DFT**

  ➡ Signal flowgraph structure

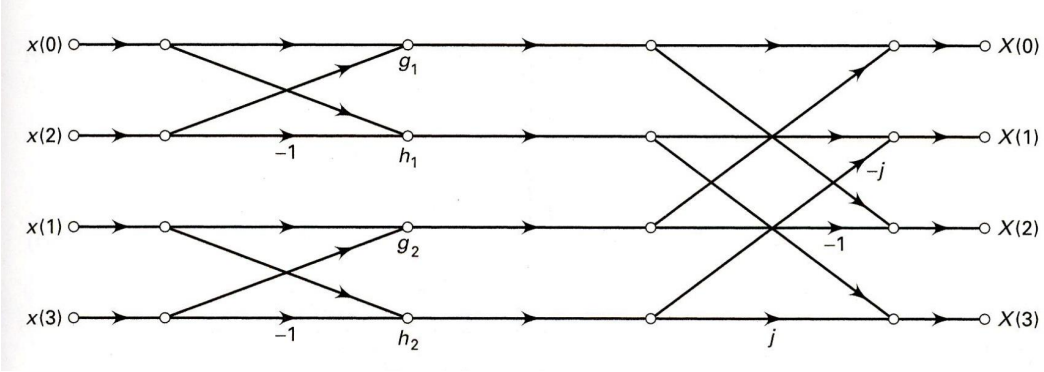
| *Step* 1 | *Step* 2 |
|---|---|
| $g_1 = x(0) + x(2)$ | $X(0) = g_1 + g_2$ |
| $g_2 = x(1) + x(3)$ | $X(1) = h_1 - jh_2$ |
| $h_1 = x(0) - x(2)$ | $X(2) = g_1 - g_2$ |
| $h_2 = x(1) - x(3)$ | $X(3) = h_1 + jh_2$ |



# Example

- **4-point DFT**

➡ The process can be interpreted differently




**1. First, a 4-point sequence x(n) is divided into two 2-point sequences.**

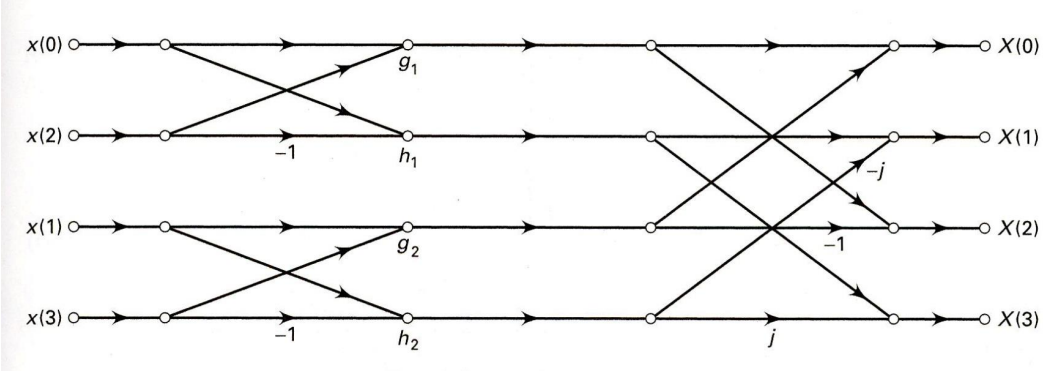
even odd

$$\left[\begin{bmatrix} x(0) \\ x(2) \end{bmatrix}, \begin{bmatrix} x(1) \\ x(3) \end{bmatrix}\right] = \begin{bmatrix} x(0) & x(1) \\ x(2) & x(3) \end{bmatrix}$$

# Example

- **4-point DFT**

➡ The process can be interpreted differently




**2. Second, a smaller 2-point DFT of each column is taken.**

$$W_2\begin{bmatrix} x(0) & x(1) \\ x(2) & x(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}\begin{bmatrix} x(0) & x(1) \\ x(2) & x(3) \end{bmatrix}$$

$$= \begin{bmatrix} x(0)+x(2) & x(1)+x(3) \\ x(0)-x(2) & x(1)-x(3) \end{bmatrix} = \begin{bmatrix} g_1 & g_2 \\ h_1 & h_2 \end{bmatrix}$$

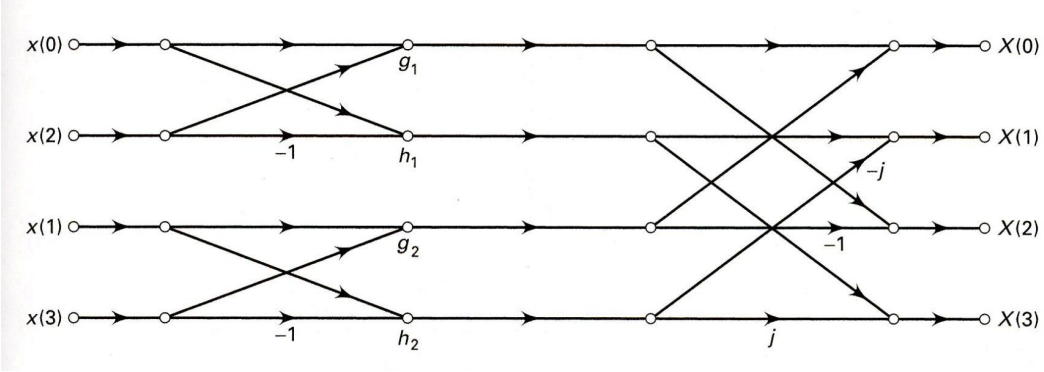
Each element is multiplied by $\{W_4^{pq}\}$ *p: row, q: column*

$$\begin{bmatrix} 1 & 1 \\ 1 & -j \end{bmatrix} .* \begin{bmatrix} g_1 & g_2 \\ h_1 & h_2 \end{bmatrix} = \begin{bmatrix} g_1 & g_2 \\ h_1 & -jh_2 \end{bmatrix}$$

# Example

- **4-point DFT**

  ➡ The process can be interpreted differently




**3. Finally, two more smaller 2-point DFT are taken of row vectors.**

$$\begin{bmatrix} g_1 & g_2 \\ h_1 & -jh_2 \end{bmatrix} W_2 = \begin{bmatrix} g_1 & g_2 \\ h_1 & -jh_2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} g_1 + g_2 & g_1 - g_2 \\ h_1 - jh_2 & h_1 + jh_2 \end{bmatrix}$$

$$= \begin{bmatrix} X(0) & X(2) \\ X(1) & X(3) \end{bmatrix}$$

*suggests a systematic approach of computing a larger DFT based on smaller DFTs.*

# Fast Fourier Transform

- **FFT**

$$X(k)=\sum_{n=0}^{N-1}x(n)W_N^{nk}=\sum_{n,even}x(n)W_N^{nk}+\sum_{n,odd}x(n)W_N^{nk},\qquad N=2^{\nu}$$

$$\left.\begin{matrix}even\\odd\end{matrix}\right\}\begin{matrix}n=2m\\n=2m+1\end{matrix},\quad m=0,1,\ldots,\frac{N}{2}-1$$

$$\begin{aligned}X(k)&=\sum_{m=0}^{N/2-1}x(2m)W_N^{2mk}+\sum_{m=0}^{N/2-1}x(2m+1)W_N^{(2m+1)k}\\&=\sum_{m=0}^{N/2-1}x(2m)\left(W_N^2\right)^{mk}+W_N^k\sum_{m=0}^{N/2-1}x(2m+1)\left(W_N^2\right)^{mk}\\&=\sum_{m=0}^{N/2-1}x(2m)W_{N/2}^{mk}+W_N^k\sum_{m=0}^{N/2-1}x(2m+1)W_{N/2}^{mk}\\&=X_e(k)+W_N^kX_o(k)\end{aligned}$$

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

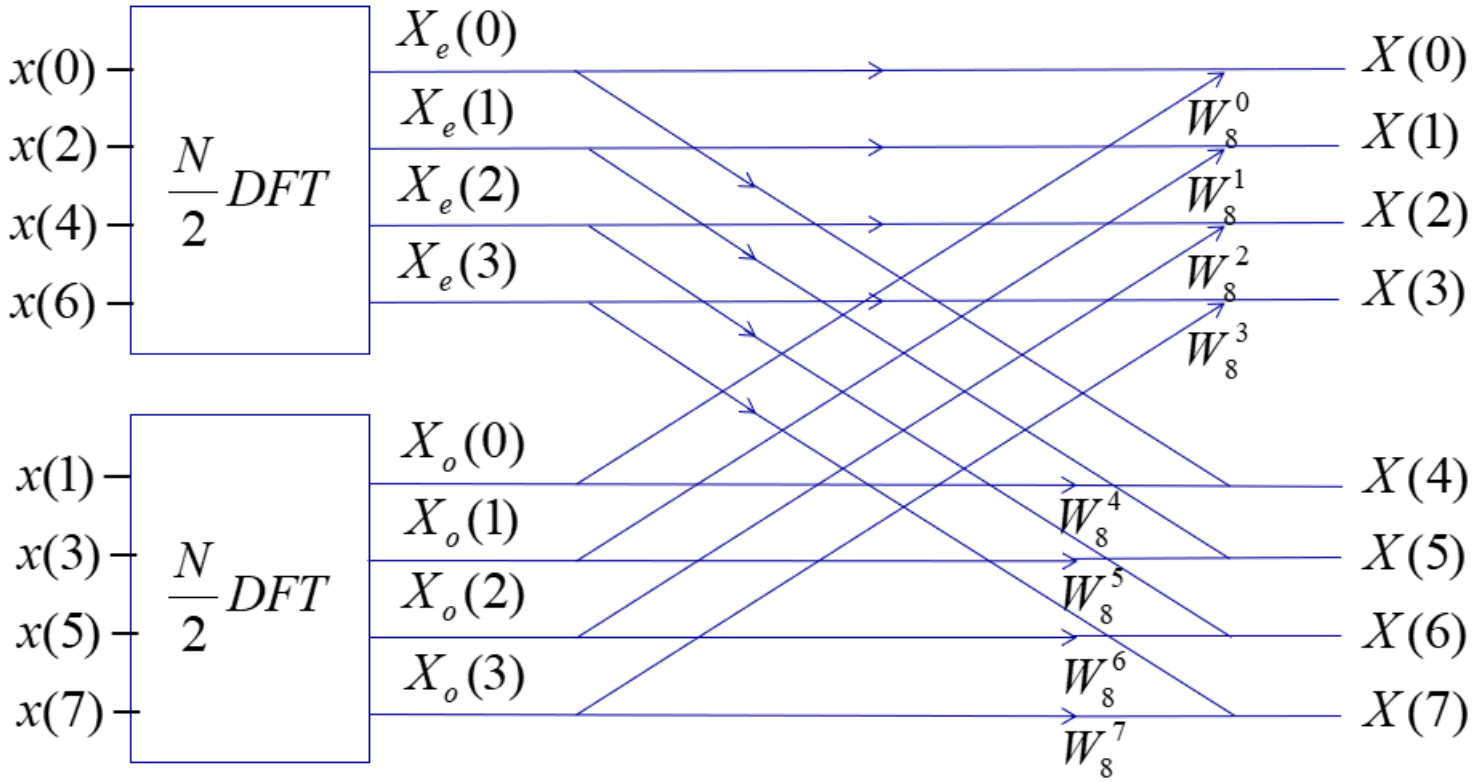
*N/2-point DFT of even samples*

*N/2-point DFT of odd samples*

# Fast Fourier Transform

- For $N = 8$

$$C_N = o\left(\frac{N^2}{2} + N\right)$$




$$2 \cdot \left(\frac{N}{2}\right)^2 + N$$

# Fast Fourier Transform

- **Further Division**

$$\frac{N}{2}: \quad 2\left(\frac{N}{2}\right)^2 + N = \frac{N^2}{2} + N$$

$$\frac{N}{2^2}: \quad 2\left(2\left(\frac{N}{2^2}\right)^2 + \frac{N}{2}\right) + N = \frac{N^2}{2^2} + 2N$$

$$\frac{N}{2^3}: \quad 2\left(2\left(2\left(\frac{N}{2^3}\right)^2 + \frac{N}{2^2}\right) + \frac{N}{2}\right) + N = \frac{N^2}{2^3} + 3N$$

$$\vdots$$

$$\frac{N}{2^\nu} = 1: \quad \frac{N^2}{2^\nu} + \nu \cdot N = \frac{N^2}{N} + \left(\log_2 N\right) \cdot N$$

# DFT vs FFT

- **Complexity**

| $N$ | $10^3$ | $10^6$ | $10^9$ |
|---|---|---|---|
| $N^2$ | $10^6$ | $10^{12}$ | $10^{18}$ |
| $N(\log_2 N)$ | $10^4$ | $20 \cdot 10^6$ | $30 \cdot 10^9$ |

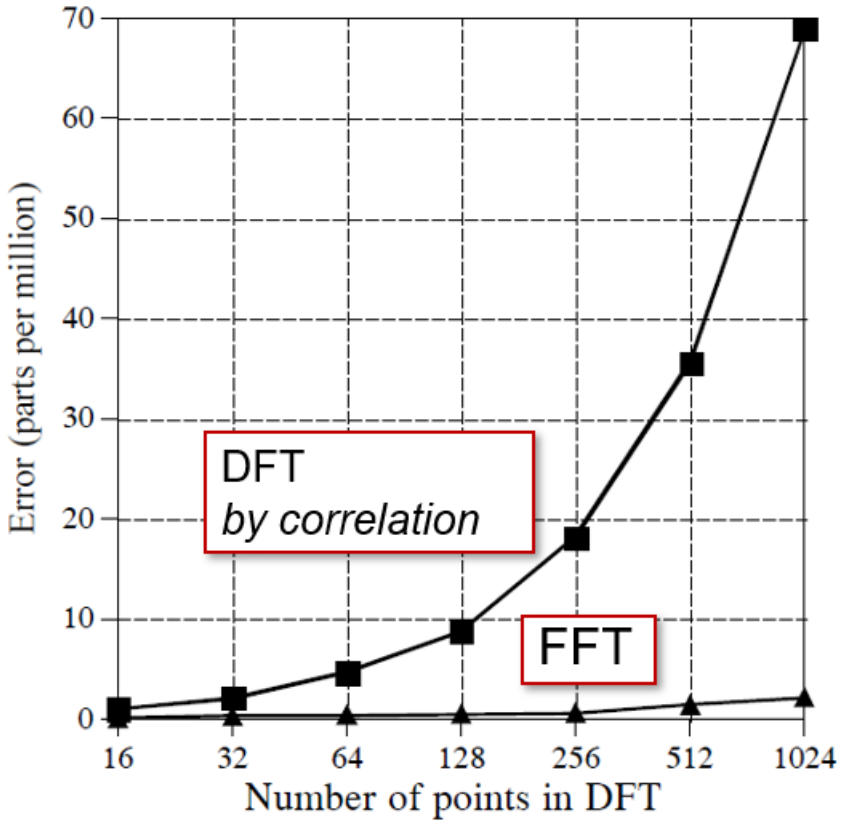
$$\log_2 10 = \frac{\log_{10} 10}{\log_{10} 2} = \frac{1}{0.3010} = 3.322$$

$$10^{18} \cdot 10^{-9} \text{ sec} = 10^9 \text{ sec} \approx 31.71 yr$$

$$30 \cdot 10^9 \cdot 10^{-9} \text{ sec} \approx 30 \text{ sec}$$

# DFT vs FFT

- **Performance**




The FFT is calculated more precisely
because the fewer number of calculations results in less round-off error.

# General Algorithm

# Divide-and-Conquer Approach

- **Basic Idea**

To reduce the DFT computation's quadratic dependence on N, one must choose a composite number N = LM

$$L^2 + M^2 << N^2 \quad for\ large\ N$$

Divide the sequence into M smaller sequence of length L,
Take M smaller L-point DFTs,
Combine these into a larger DFT using L smaller M-point DFTs.

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

$$n = Ml + m, \quad 0 \le l \le L-1, \quad 0 \le m \le M-1$$

$$k = p + Lq, \quad 0 \le p \le L-1, \quad 0 \le q \le M-1$$

# Divide-and-Conquer Approach

- **One dimensional**

*One dimensional data array for storing the sequence* $x(n),\ 0 \le n \le N-1.$

| $n \rightarrow$ | 0 | 1 | 2 | $\cdots$ | $N-1$ |
|---|---|---|---|---|---|
| | $x(0)$ | $x(1)$ | $x(2)$ | $\cdots$ | $x(N-1)$ |

- **Two dimensional**

*Two dimensional data array for storing the sequence* $x(n),\ 0 \le n \le N-1.$

| | $m$ *column index* | | | | |
|---|---|---|---|---|---|
| *row index* $l$ | 0 | 1 | 2 | $\cdots$ | $M-1$ |
| 0 | $x(0,0)$ | $x(0,1)$ | | $\cdots$ | |
| 1 | $x(1,0)$ | $x(1,1)$ | | $\cdots$ | |
| 2 | $x(2,0)$ | $x(2,1)$ | | $\cdots$ | |
| $\vdots$ | $\vdots$ | $\vdots$ | | $\cdots$ | $\vdots$ |
| $L-1$ | | | | $\cdots$ | |

# Divide-and-Conquer Approach

- **Row-wise**

$$\textit{Row-wise} \quad n = Ml + m$$

| $l$ \ $m$ | 0 | 1 | 2 | $\cdots$ | $M-1$ |
|---|---|---|---|---|---|
| 0 | $x(0)$ | $x(1)$ | $x(2)$ | $\cdots$ | $x(M-1)$ |
| 1 | $x(M)$ | $x(M+1)$ | $x(M+2)$ | $\cdots$ | $x(2M-1)$ |
| 2 | $x(2M)$ | $x(2M+1)$ | $x(2M+2)$ | $\cdots$ | $x(3M-1)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | | $\cdots$ | $\vdots$ |
| $L-1$ | $x(L-1)M$ | $x((L-1)M+1)$ | $x((L-1)M+2)$ | $\cdots$ | $x(LM-1)$ |

- **Column-wise**

$$\textit{Column-wise} \quad n = l + mL$$

| $l$ \ $m$ | 0 | 1 | 2 | $\cdots$ | $M-1$ |
|---|---|---|---|---|---|
| 0 | $x(0)$ | $x(L)$ | $x(2L)$ | $\cdots$ | $x((M-1)L)$ |
| 1 | $x(1)$ | $x(L+1)$ | $x(2L+1)$ | $\cdots$ | $x((M-1)L+1)$ |
| 2 | $x(2)$ | $x(L+2)$ | $x(2L+2)$ | $\cdots$ | $x((M-1)L+2)$ |
| $\vdots$ | $\vdots$ | $\vdots$ | | $\cdots$ | $\vdots$ |
| $L-1$ | $x(L-1)$ | $x(2L-1)$ | $x(3L-1)$ | $\cdots$ | $x(LM-1)$ |

# Divide-and-Conquer Approach

- **Definition**

  x(n) is mapped into the rectangular array x(l,m) and
  X(k) is mapped into a corresponding rectangular array X(p,q).

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

$$n=Ml+m,\quad 0\le l\le L-1,\quad 0\le m\le M-1$$

$$k=p+Lq,\quad 0\le p\le L-1,\quad 0\le q\le M-1$$

Row-wise, n=Ml+m

Column-wise, k=p+Lq

$$X(p,q)=\sum_{m=0}^{M-1}\sum_{l=0}^{L-1}x(l,m)W_N^{(Ml+m)(p+Lq)}$$

$$W_N^{Mlp}=W_{N/M}^{lp}=W_L^{lp},W_N^{MLlq}=W_N^{Nlq}=1,W_N^{Lmq}=W_{N/L}^{mq}=W_M^{mq}$$

$$=\sum_{m=0}^{M-1}\left\{W_N^{mp}\left[\sum_{l=0}^{L-1}x(l,m)W_N^{Mlp}\right]\right\}W_N^{Lmq}$$

$$=\underbrace{\sum_{m=0}^{M-1}\left\{W_N^{mp}\underbrace{\left[\sum_{l=0}^{L-1}x(l,m)W_L^{lp}\right]}_{L-\text{point DFT}}\right\}W_M^{mq}}_{M-\text{point DFT}}$$

# Divide-and-Conquer Approach

- **Definition**

  x(n) is mapped into the rectangular array x(l,m) and
  X(k) is mapped into a corresponding rectangular array X(p,q).

  *Column-wise, n=mL+l*

  *row-wise, k=Mp+q*

$$X(p,q)=\sum_{m=0}^{M-1}\sum_{l=0}^{L-1}x(l,m)W_N^{(Mp+q)(mL+l)}$$

$$=\sum_{l=0}^{L-1}\left\{W_N^{lq}\left[\sum_{m=0}^{M-1}x(l,m)W_M^{mq}\right]\right\}W_L^{lp}$$

$$W_N^{MLmp}=W_N^{Nmp}=1, W_N^{mqL}=W_{N/L}^{mq}=W_M^{mq}, W_N^{Mpl}=W_{N/M}^{pl}=W_L^{pl}$$

*The computational complexity is*
*Complex multiplication:* $N(L+M+1)$
*Complex addition:* $N(L+M-2)$

# Divide-and-Conquer Approach

- **Algorithm1**

1. Compute the $L$-point $DFT$ array

$$F(p,m)=\sum_{l=0}^{L-1}x(l,m)W_L^{lp};\quad 0\le p\le L-1$$

for each of the columns $m=0,\ldots,M-1$.

$M$ $DFTs$, each of $L$ points

$ML^2$ multiplications

$ML(L-1)$ additions

2. Modify $F(p,m)$ to obtain another array

$$G(p,m)=W_N^{pm}F(p,m),\quad \begin{matrix}0\le p\le L-1\\ 0\le m\le M-1\end{matrix}$$

$LM$ multiplications

3. Compute the $M$-point $DFTs$

$$X(p,q)=\sum_{m=0}^{M-1}G(p,m)W_M^{mq},\quad 0\le q\le M-1$$

for each of the rows $p=0,\ldots,L-1$.

$LM^2$ multiplications

$LM(M-1)$ additions

# Divide-and-Conquer Approach

- **Algorithm2**

1. Compute the $M$-point $DFT$ array

$$F(l,q)=\sum_{m=0}^{M-1}x(l,m)W_M^{mq};\quad 0\le q\le M-1$$

for each of the rows $l=0,\ldots,L-1$.

$L$ *DFTs, each of* $M$ *points*

$LM^2$ *multiplications*

$LM(M-1)$ *additions*

2. Modify $F(l,q)$ to obtain another array

$$G(l,q)=W_N^{lq}F(l,q),\quad \begin{matrix}0\le l\le L-1\\ 0\le q\le M-1\end{matrix}$$

$LM$ *multiplications*

3. Compute the $L$-point $DFTs$

$$X(p,q)=\sum_{l=0}^{L-1}G(l,q)W_L^{lp},\quad 0\le p\le M-1$$

for each of the columns $q=0,\ldots,M-1$.

$ML^2$ *multiplications*

$ML(L-1)$ *additions*

# Example: Algorithm2

Compute the 15-point *DFT*.

$$N = 5 \times 3 = 15, \quad L = 5, M = 3$$

*Store the* 15*-point sequence* $x(n)$ *column-wise*

| | | | |
|---|---|---|---|
| Row 1: | $x(0,0)=x(0)$ | $x(0,1)=x(5)$ | $x(0,2)=x(10)$ |
| Row 2: | $x(1,0)=x(1)$ | $x(1,1)=x(6)$ | $x(1,2)=x(11)$ |
| Row 3: | $x(2,0)=x(2)$ | $x(2,1)=x(7)$ | $x(2,2)=x(12)$ |
| Row 4: | $x(3,0)=x(3)$ | $x(3,1)=x(8)$ | $x(3,2)=x(13)$ |
| Row 5: | $x(4,0)=x(4)$ | $x(4,1)=x(9)$ | $x(4,2)=x(14)$ |

*Compute the* 3*-point DFTs for each of the* 5 *rows*

$$F(l,q) = \sum_{m=0}^{M-1} x(l,m) W_M^{mq};$$

| | | |
|---|---|---|
| $F(0,0)$ | $F(0,1)$ | $F(0,2)$ |
| $F(1,0)$ | $F(1,1)$ | $F(1,2)$ |
| $F(2,0)$ | $F(2,1)$ | $F(2,2)$ |
| $F(3,0)$ | $F(3,1)$ | $F(3,2)$ |
| $F(4,0)$ | $F(4,1)$ | $F(4,2)$ |

# Example: Algorithm2

*Multiply each of the term $F(l,q)$ by the phase factor $W_N^{lq} = W_{15}^{lq}$,*

$$0 \le l \le 4, \quad 0 \le q \le 2$$

$$G(l,q) = W_N^{lq} F(l,q),$$

$$\begin{matrix} G(0,0) & G(0,1) & G(0,2) \\ G(1,0) & G(1,1) & G(1,2) \\ G(2,0) & G(2,1) & G(2,2) \\ G(3,0) & G(3,1) & G(3,2) \\ G(4,0) & G(4,1) & G(4,2) \end{matrix}$$

*Compute the 5-point DFTs for each of the 3 columns*

$$X(p,q) = \sum_{l=0}^{L-1} G(l,q) W_L^{lp}$$

$$\begin{matrix} X(0,0)=X(0) & X(0,1)=X(1) & X(0,2)=X(2) \\ X(1,0)=X(3) & X(1,1)=X(4) & X(1,2)=X(5) \\ X(2,0)=X(6) & X(2,1)=X(7) & X(2,2)=X(8) \\ X(3,0)=X(9) & X(3,1)=X(10) & X(3,2)=X(11) \\ X(4,0)=X(12) & X(4,1)=X(13) & X(4,2)=X(14) \end{matrix}$$

# Example: Algorithm2

INPUT ARRAY

$x(0)\ x(5)\ x(10)\ x(1)\ x(6)\ x(11)\ x(2)\ x(7)\ x(12)\ x(3)\ x(8)\ x(13)\ x(4)\ x(9)\ x(14)$

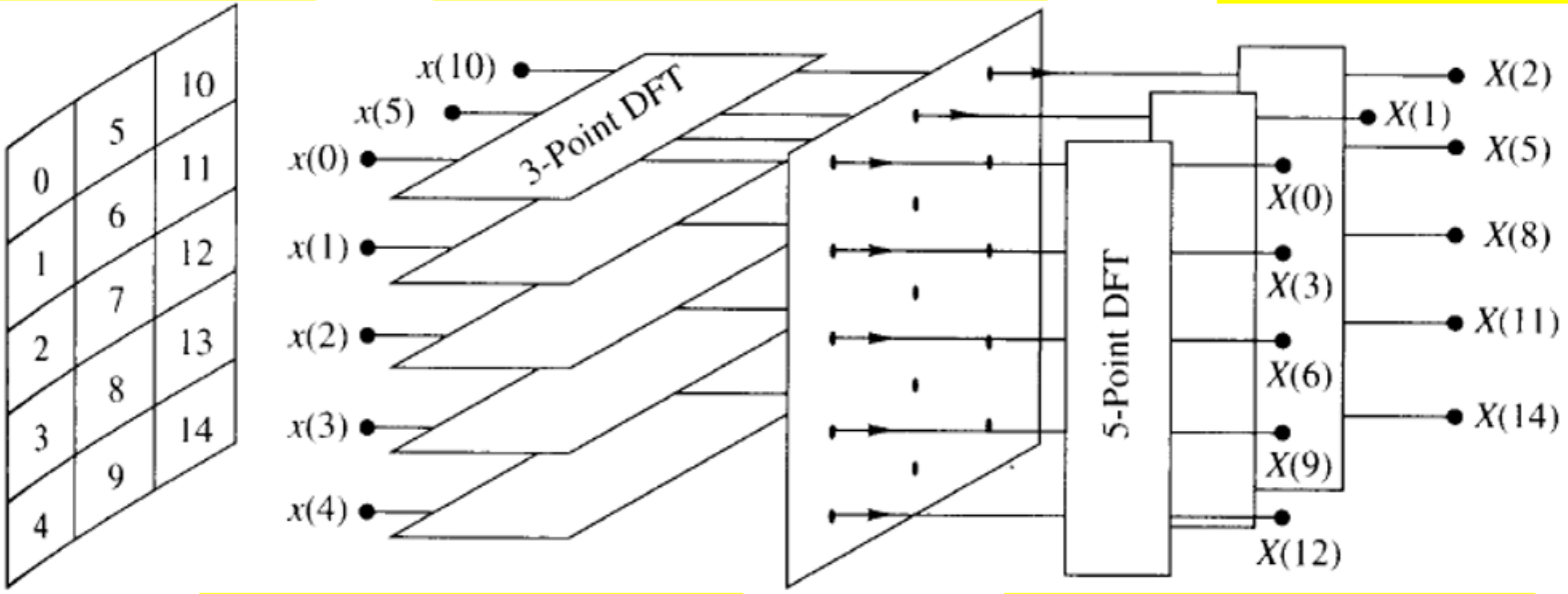
OUTPUT ARRAY

$X(0)\ X(1)\ X(2)\ X(3)\ X(4)\ X(5)\ X(6)\ X(7)\ X(8)\ X(9)\ X(10)\ X(11)\ X(12)\ X(13)\ X(14)$

*Store the signal column-wise*

*Multiply the resulting array by the phase factors* $W_N^{lq}$ $W_{15}^{lq}$

*Read the resulting array row-wise*




*Compute the M-point DFT of each row*

*Compute the L-point DFT of each column*

# Radix of FFT

The divide-and-combine approach is applicable when the number N of data points is not a prime.

$$N = r_1 r_2 r_3 \cdots r_\nu$$

*factored into a product of prime numbers, decomposition repeated* $(\nu - 1)$ *more times.*

Of particular importance is the case in which

$$r_1 = r_2 = \cdots = r_\nu = r, \quad N = r^\nu$$

The radix of the FFT algorithm

The DFTs are of size r,
so that the computation of the N-point DFT has a regular pattern.

# Example: Radix-2 FFT

$$N = 2^{\nu};\quad M = 2, L = N/2$$

*Divide* $x(n)$ *into two* $N/2$*-point sequences* $f_1(n), f_2(n)$

*contains even-ordered samples of* $x(n)$ $\quad f_1(n) = x(2n)$

*contains odd-ordered samples of* $x(n)$ $\quad f_2(n) = x(2n+1)$

$$; \quad 0 \le n \le \frac{N}{2} - 1$$

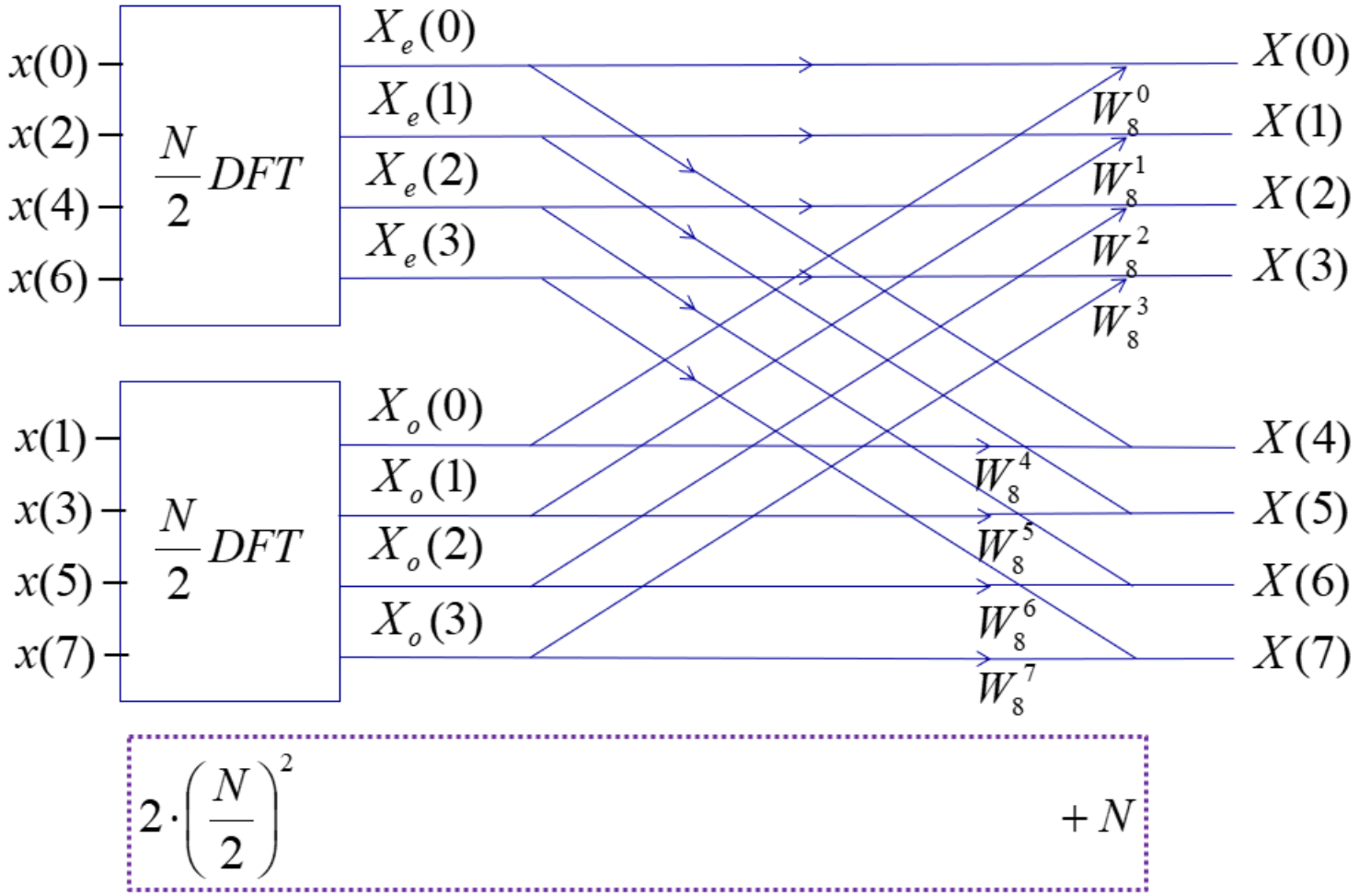
$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \quad k = 0, 1, \ldots, N-1$$

$$= \sum_{n,even} x(n) W_N^{kn} + \sum_{n,odd} x(n) W_N^{kn}$$

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

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

# Example: Radix-2 FFT



# Example: Radix-4 FFT

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

$$= \sum_{n=0}^{N/4-1} x(n) W_N^{nk} + \sum_{n=N/4}^{2N/4-1} x(n) W_N^{nk} + \sum_{n=2N/4}^{3N/4-1} x(n) W_N^{nk} + \sum_{n=3N/4}^{N-1} x(n) W_N^{nk}$$

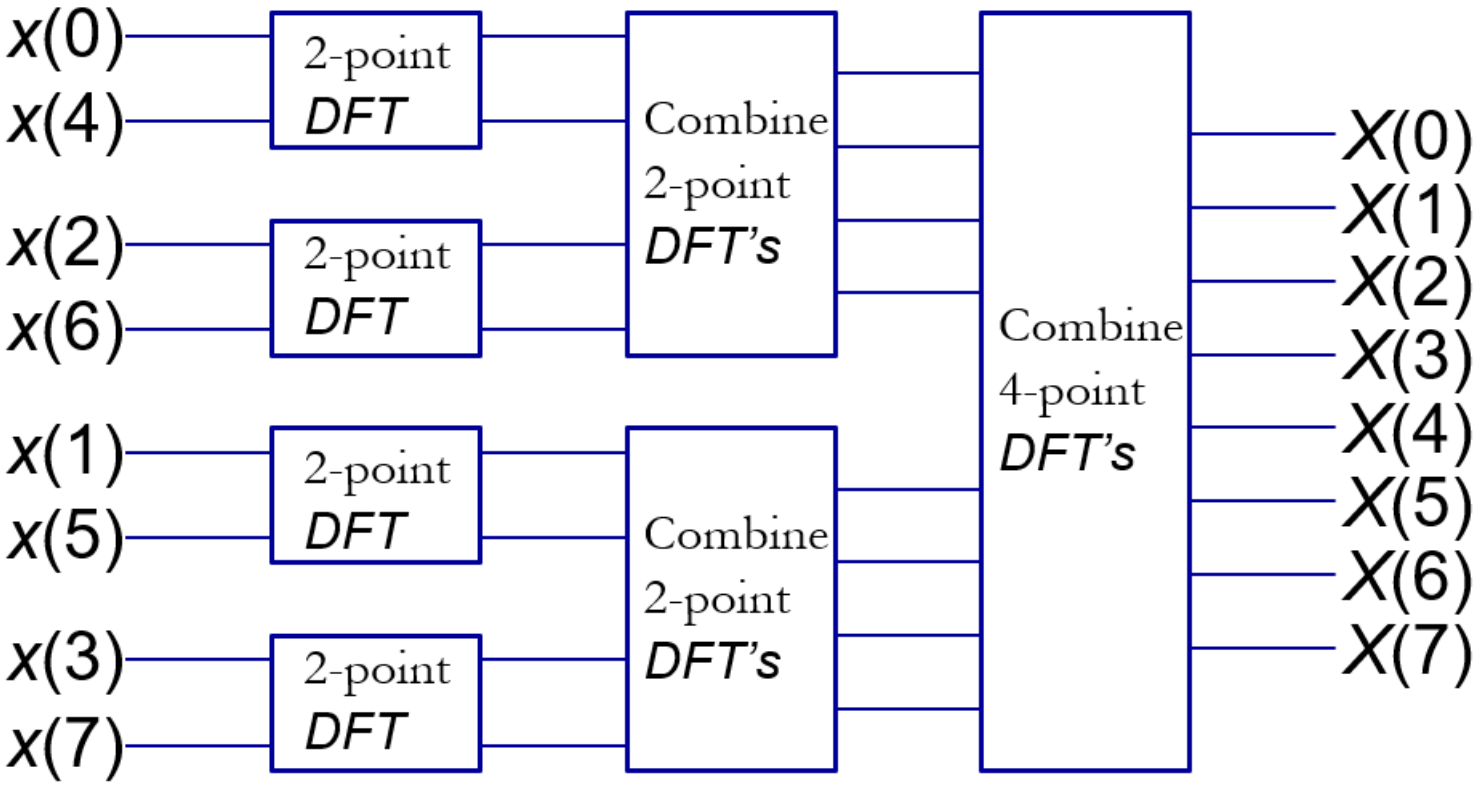
$$= \sum_{n=0}^{N/4-1} x(n) W_N^{nk} + \sum_{n=0}^{N/4-1} x\left(n+N/4\right) W_N^{(n+N/4)k} + \sum_{n=0}^{N/4-1} x\left(n+N/2\right) W_N^{(n+N/2)k}$$
$$+ \sum_{n=0}^{N/4-1} x\left(n+3N/4\right) W_N^{(n+3N/4)k}$$

$$= \sum_{n=0}^{N/4-1} \left[ \begin{array}{l} x(n) + x\left(n+N/4\right) W_N^{(n+N/4)k} + x\left(n+N/2\right) W_N^{(n+N/2)k} \\ + x\left(n+3N/4\right) W_N^{(n+N/2)k} \end{array} \right] W_N^{nk}$$

# Example: Radix-4 FFT

*The decimation of the data sequence can be repeated again and again until the resulting sequences are reduced to* 1*-point sequences.*

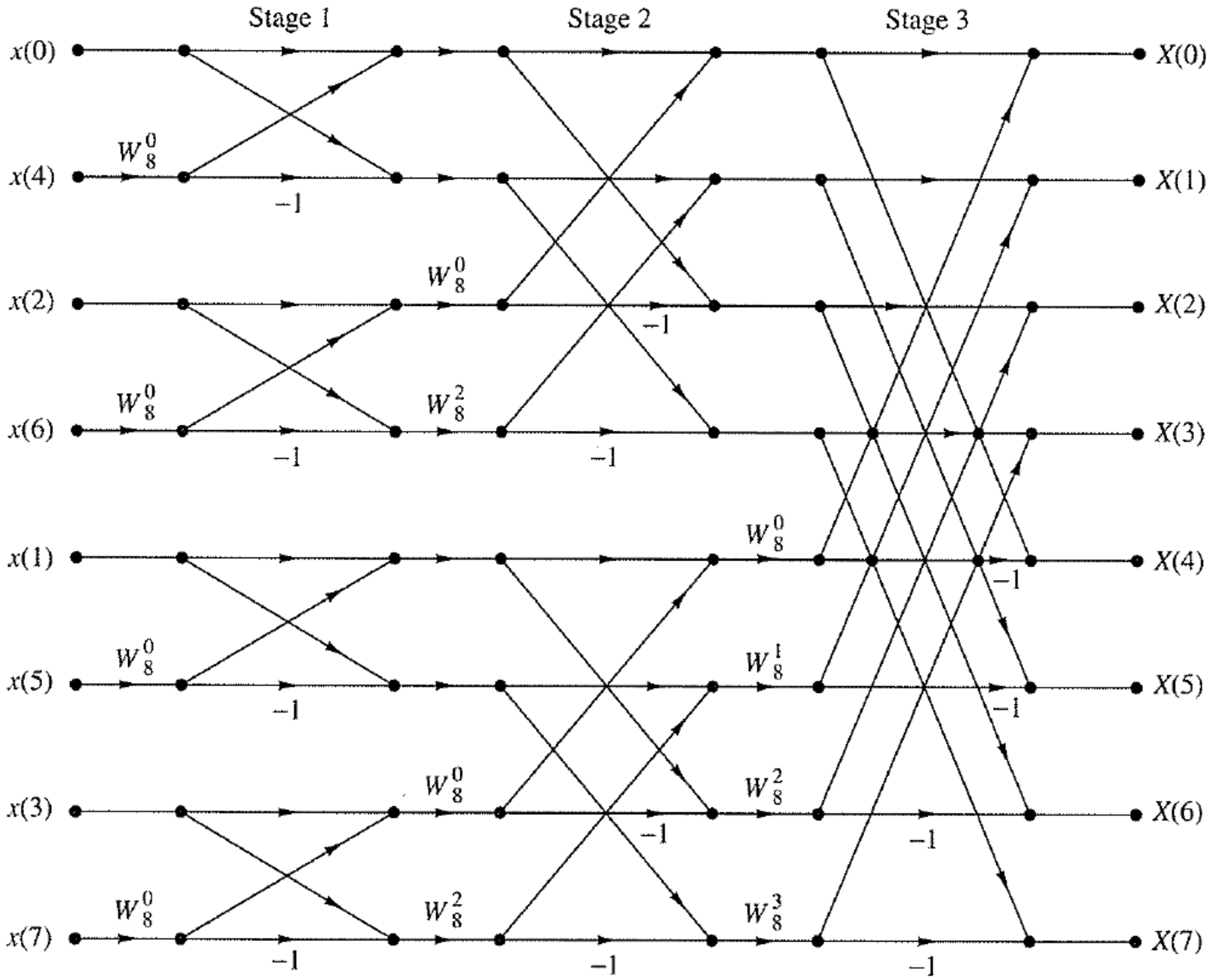
$$N = 2^{\nu} \rightarrow \nu = \log_2 N \text{ times} \qquad c_N = (N/2)\log_2 N$$




*Three stages in the computation of an* N=8*-point DFT*

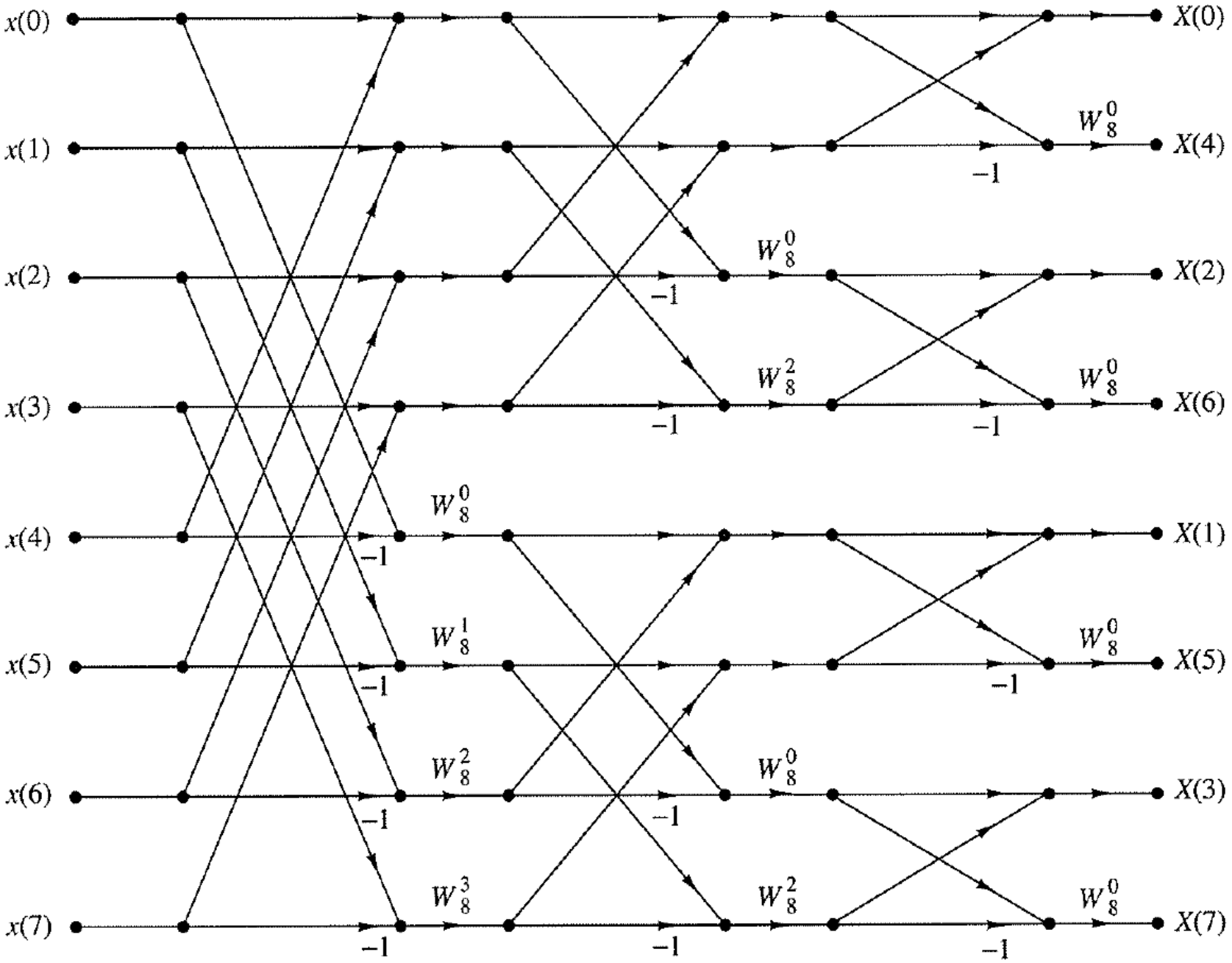
# *Decimation-in-time FFT structure for* $N = 8$

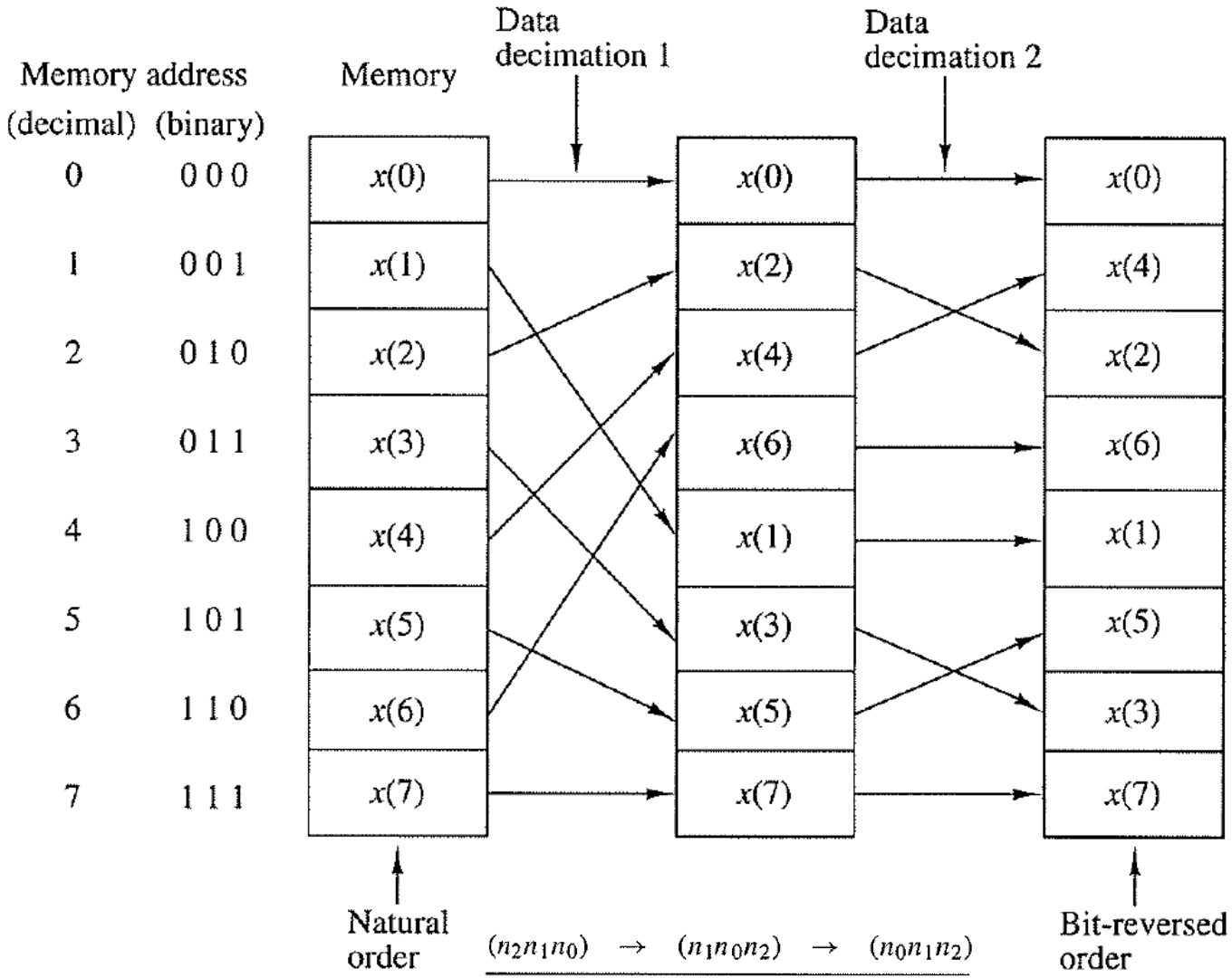
$$C_N = N\nu = N \log_2 N$$



# *Decimation-in-frequency* FFT *structure for* $N = 8$

$C_N = N\nu = N \log_2 N$

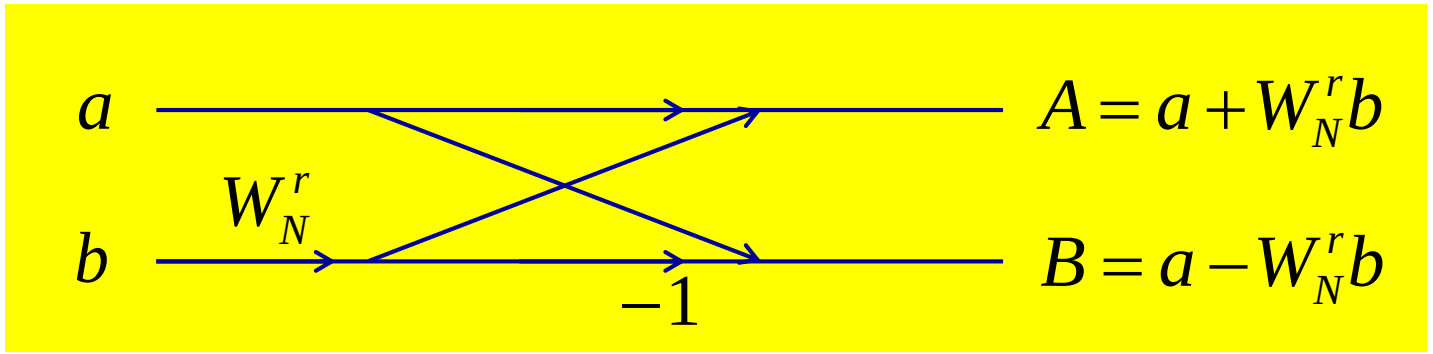




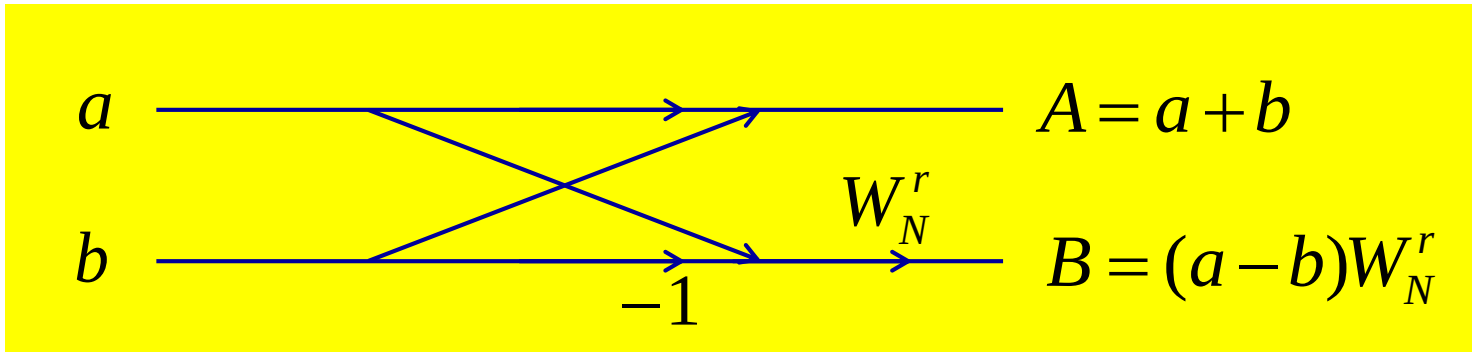

| $(n_2n_1n_0)$ | → | $(n_1n_0n_2)$ | → | $(n_0n_1n_2)$ |
|---|---|---|---|---|
| (0 0 0) | → | (0 0 0) | → | (0 0 0) |
| (0 0 1) | → | (0 1 0) | → | (1 0 0) |
| (0 1 0) | → | (1 0 0) | → | (0 1 0) |
| (0 1 1) | → | (1 1 0) | → | (1 1 0) |
| (1 0 0) | → | (0 0 1) | → | (0 0 1) |
| (1 0 1) | → | (0 1 1) | → | (1 0 1) |
| (1 1 0) | → | (1 0 1) | → | (0 1 1) |
| (1 1 1) | → | (1 1 1) | → | (1 1 1) |

$N = 2^{\nu}, \quad \nu = \log_2 N$ *stages*

$N/2$ *butterflies per stage of the computation process*

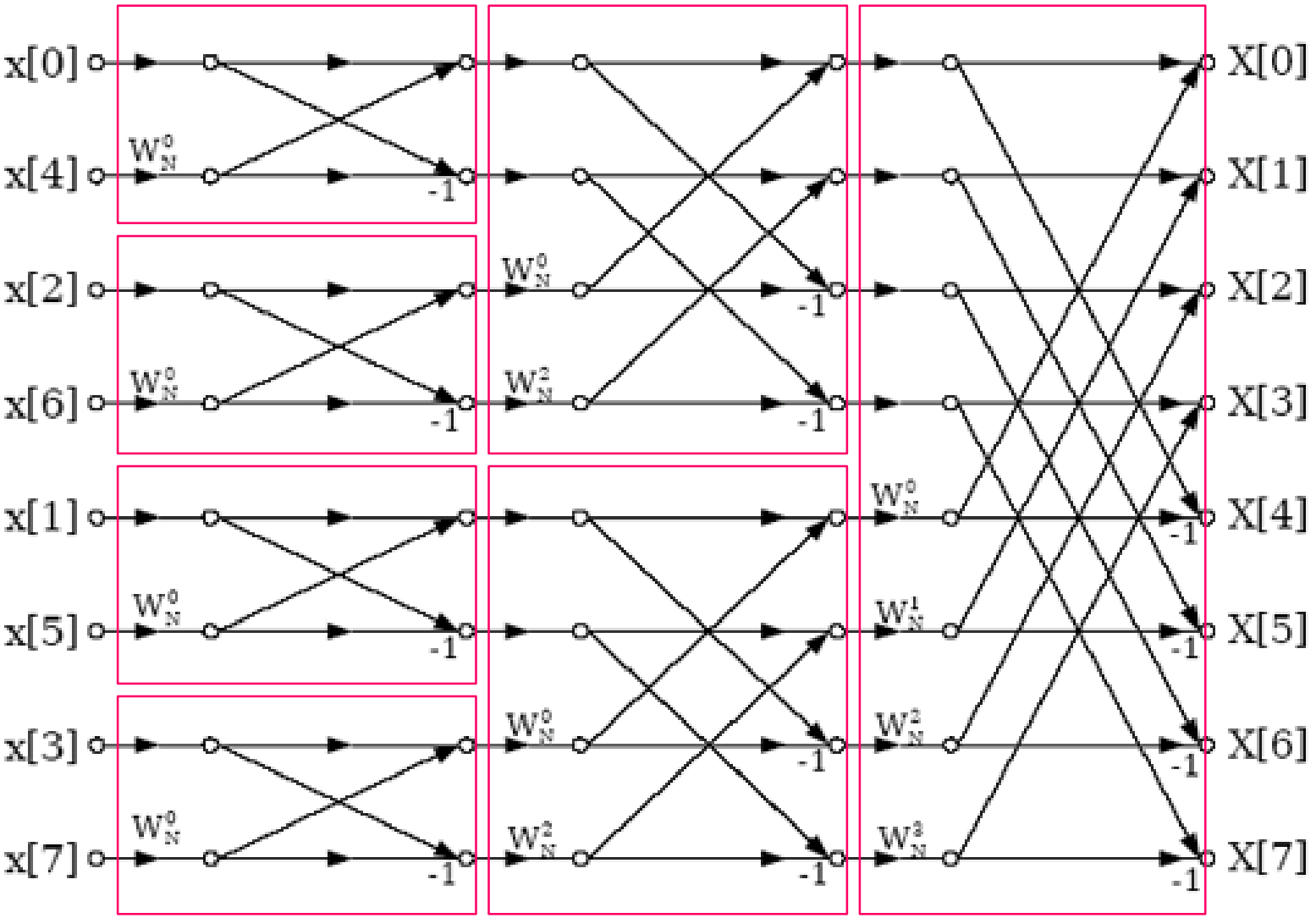



*Basic butterfly computation in the decimation-in-time FFT algorithm*




*Basic butterfly computation in the decimation-in-frequency FFT algorithm*

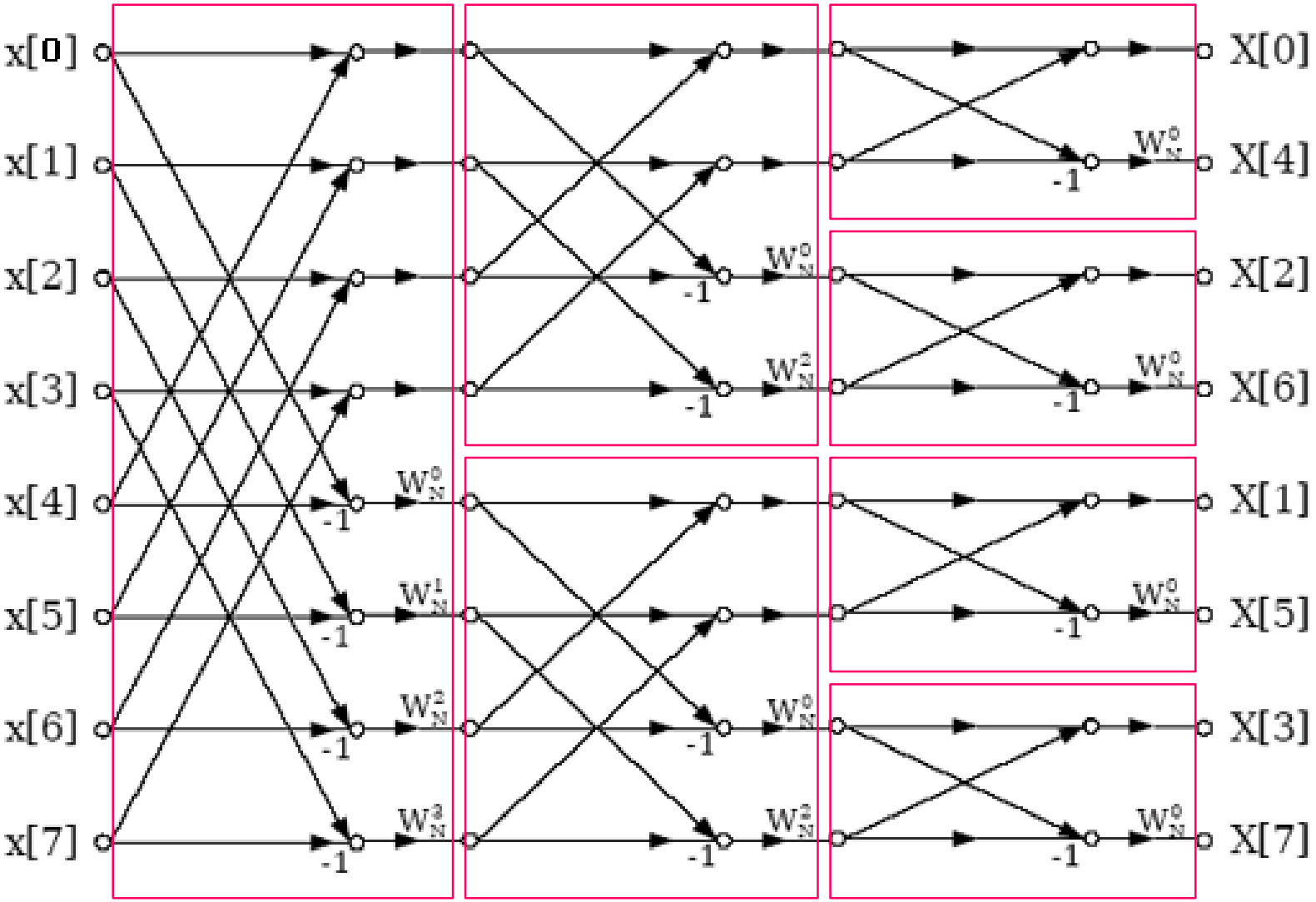
# *Decimation-in-time FFT*




*bit-reversed order*

*natural order*

# *Decimation-in-frequency* FFT




*natural order*

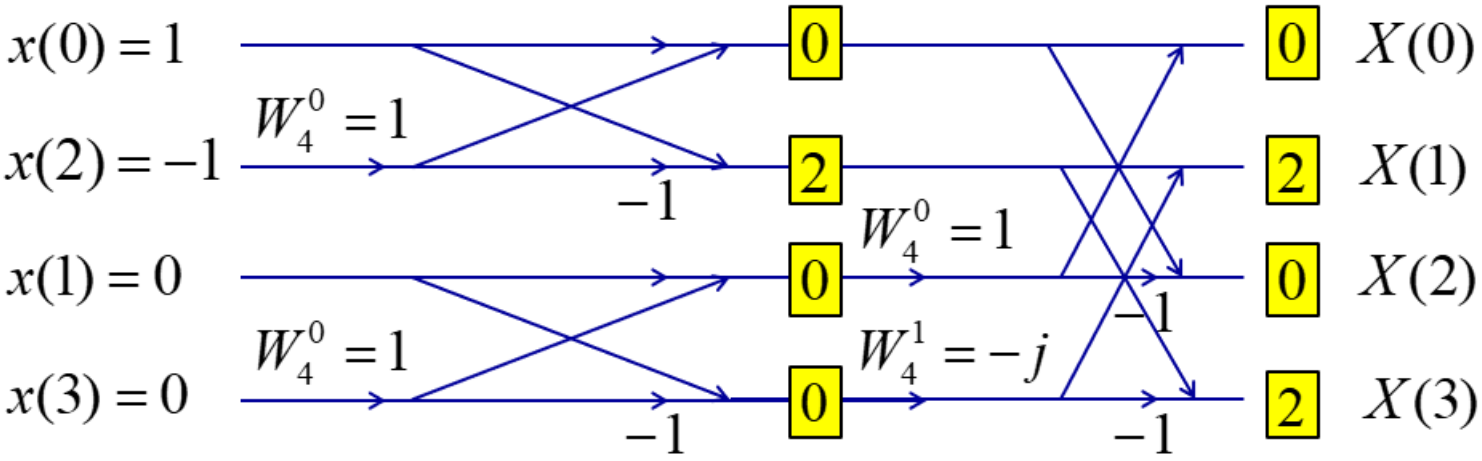
*bit-reversed order*

# Example:

*Find the* **4**-*point* **DFT** *of the sequence by using* **DIT-FFT**.

$$x(n) = \{1, 0, -1, 0\}$$

$$X(k) = \sum_{n=0}^{3} x(n) W_4^{nk}, \quad 0 \le k \le 3; \quad W_4 = e^{-j2\pi/4} = -j$$




$$X(k) = \{0, 2, 0, 2\}$$

# Homework #11.1 (2 pt.): Due Feb 9

Compute the 8-point *DFT* of the sequence

$$x(n) = \begin{cases} 1, & 0 \le n \le 7 \\ 0, & otherwise \end{cases}$$

by using *DIF-FFT*. Follow exactly the corresponding signal flow graph and keep track of all the intermediate quantities by putting them on the diagram.

# How the FFT works?

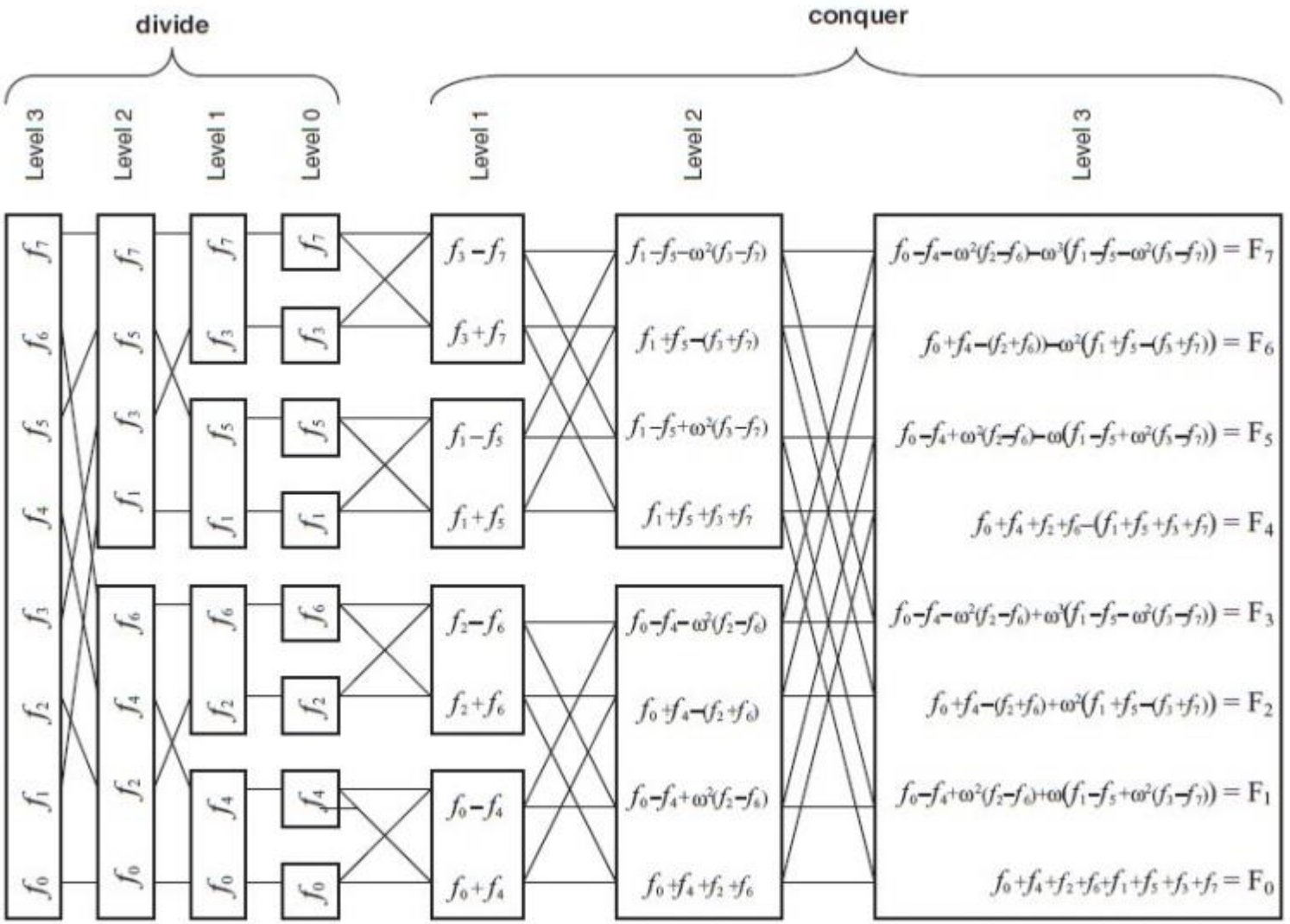
# How the FFT works?

1. Decompose an *N* point time domain signal into *N* time domain signals each composed of a single point.

2. Calculate the *N* frequency spectra corresponding to these *N* time domain signals.
   (*The frequency spectrum of a* 1 *point signal is equal to itself.*) *Nothing is required to do here!*

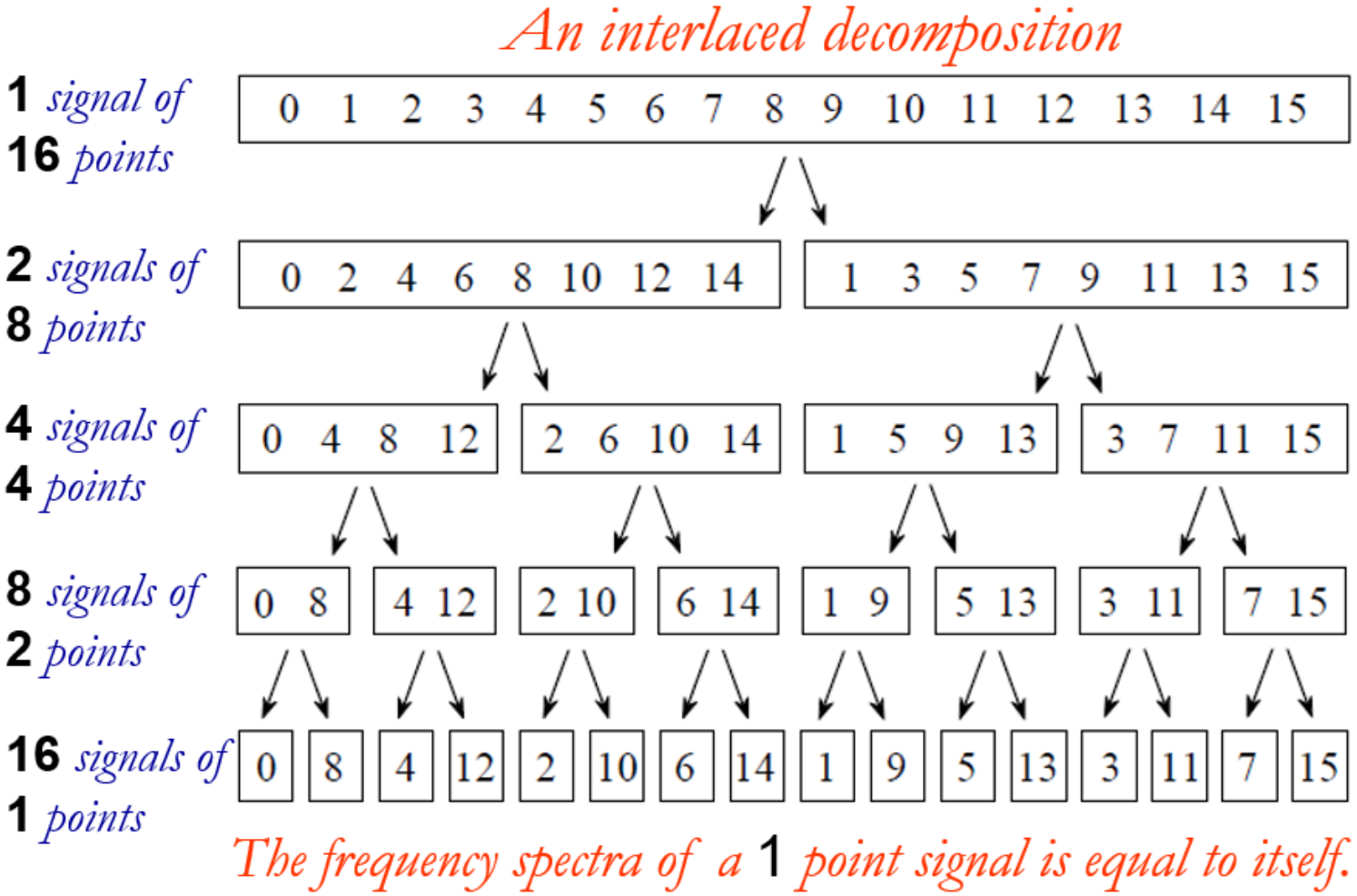
3. The *N* spectra are synthesized into a single frequency spectrum.
   (*Combine the N frequency spectra in the exact reverse order that the time domain decomposition took place.*)

# Divide-and-Conquer Approach



# Division / Decomposition

The FFT (time domain) Decomposition

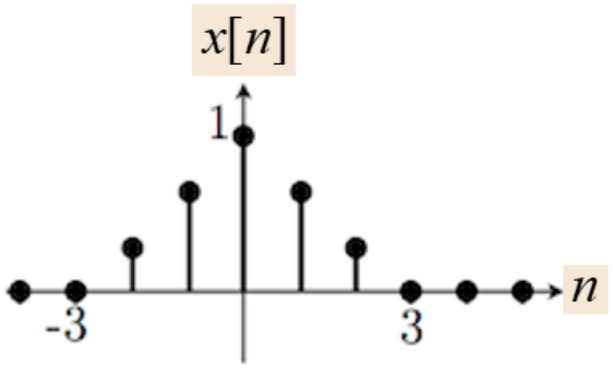


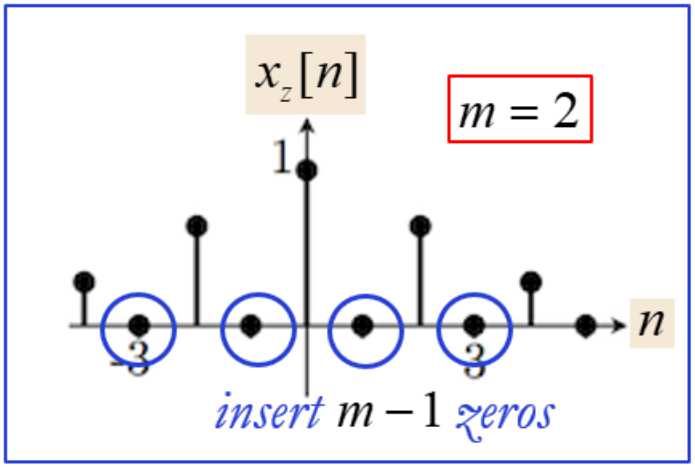
# Division / Decomposition

## The FFT Bit Reversal Sorting

| *Decimal* | *Binary* | | *Decimal* | *Binary* |
|---|---|---|---|---|
| 0 | 0000 | | 0 | 0000 |
| 1 | 0001 | | 8 | 1000 |
| 2 | 0010 | | 4 | 0100 |
| 3 | 0011 | | 12 | 1100 |
| 4 | 0100 | | 2 | 0010 |
| 5 | 0101 | | 10 | 1010 |
| 6 | 0110 | | 6 | 0100 |
| 7 | 0111 | ⇒ | 14 | 1110 |
| 8 | 1000 | | 1 | 0001 |
| 9 | 1001 | | 9 | 1001 |
| 10 | 1010 | | 5 | 0101 |
| 11 | 1011 | | 13 | 1101 |
| 12 | 1100 | | 3 | 0011 |
| 13 | 1101 | | 11 | 1011 |
| 14 | 1110 | | 7 | 0111 |
| 15 | 1111 | | 15 | 1111 |

# Conquer: FFT Synthesis



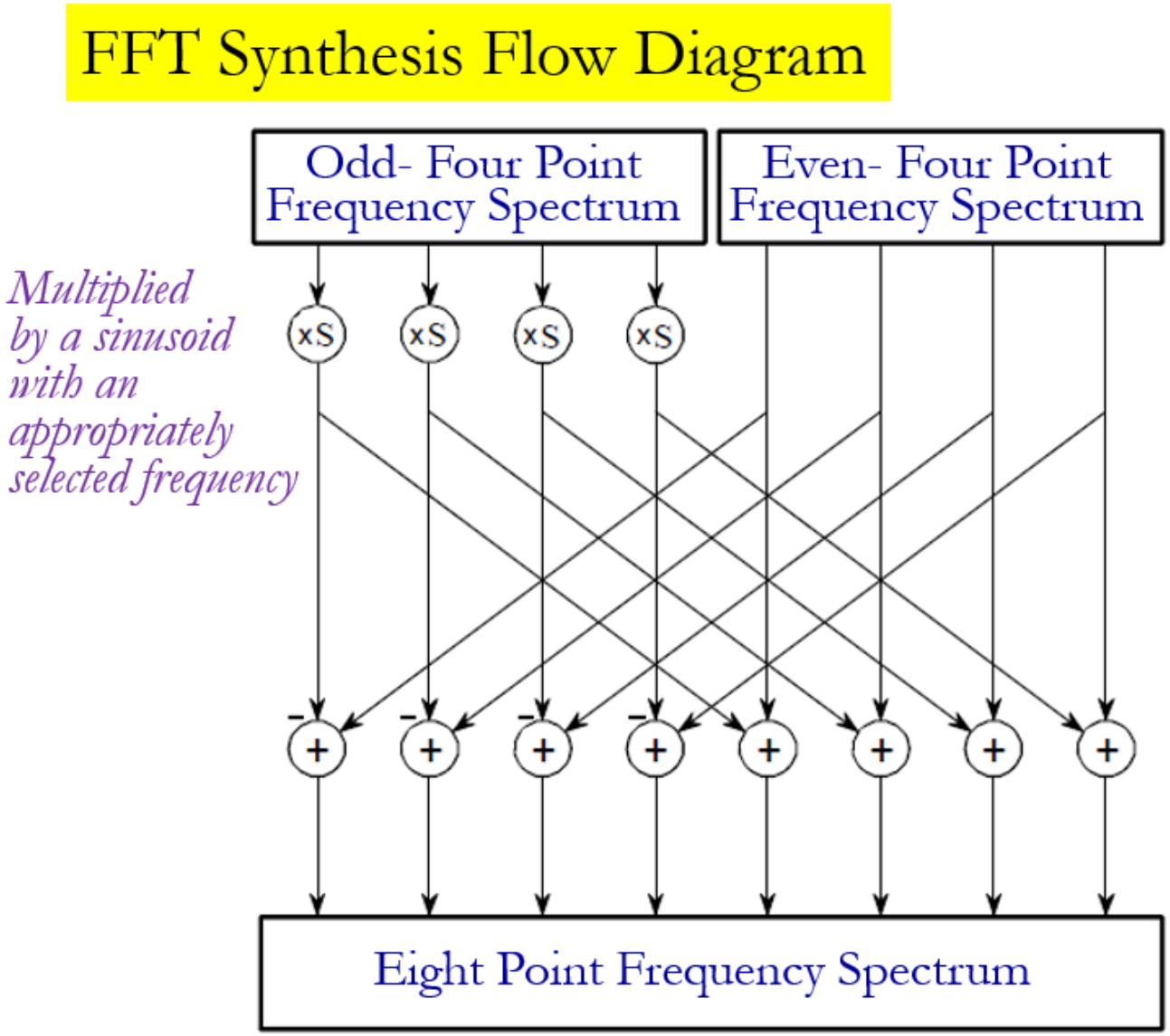




$$x_z[n] = \begin{cases} x[n/m], & n/m \text{ integer} \\ 0, & \text{otherwise} \end{cases} = \sum_{k=-\infty}^{\infty} x(k)\delta(n-km)$$

$$X_Z(e^{j\omega}) = \sum_{n=-\infty}^{\infty} \left\{ \sum_{k=-\infty}^{\infty} x(k)\delta(n-km) \right\} e^{-j\omega n}$$

$$= \sum_{k=-\infty}^{\infty} x(k) \left\{ \sum_{n=-\infty}^{\infty} \delta(n-km) e^{-j\omega n} \right\} = \sum_{k=-\infty}^{\infty} x(k) e^{-j\omega mk} = X(e^{j\omega m})$$

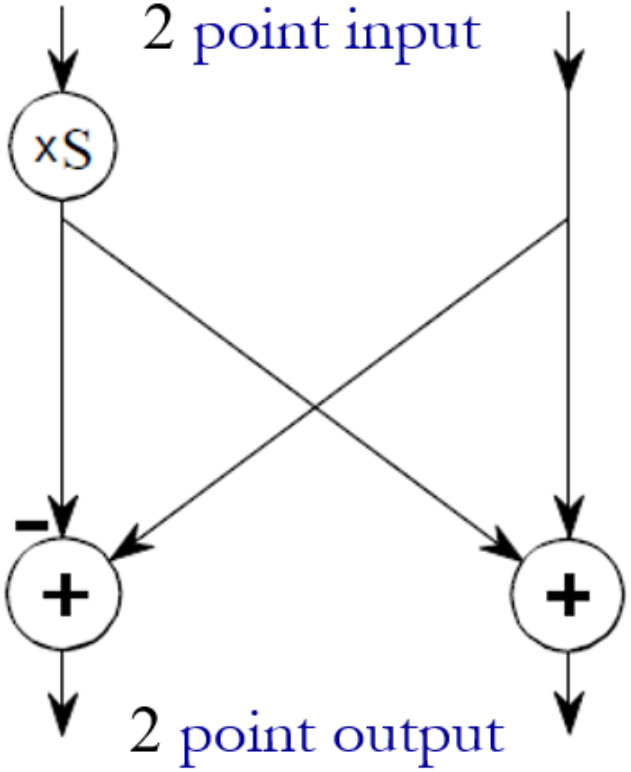
# Conquer: FFT Synthesis Flow Diagram



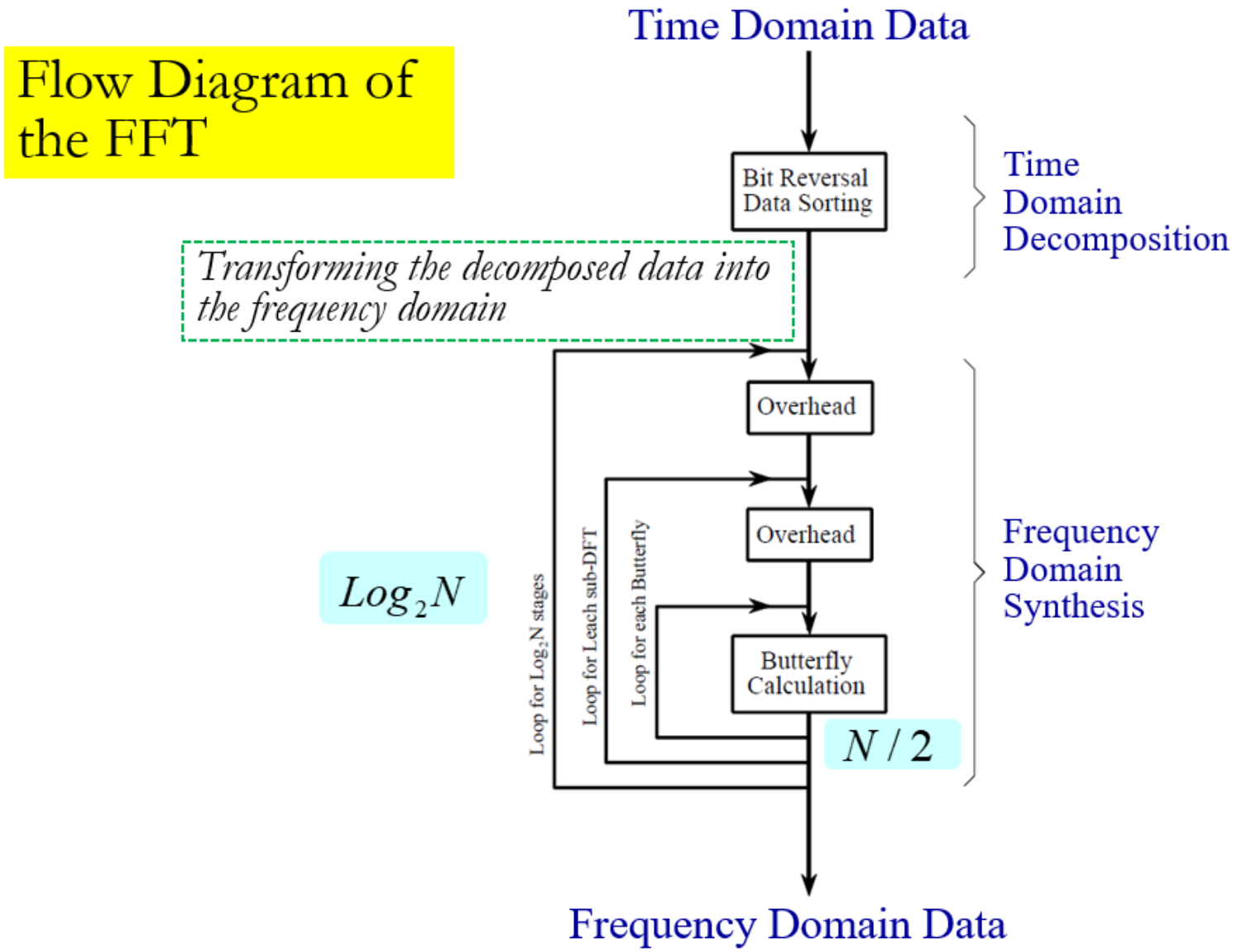
# Conquer: FFT Synthesis Flow Diagram

The FFT Butterfly

*The basic calculation element: taking two complex points and converting them into two other complex points.*



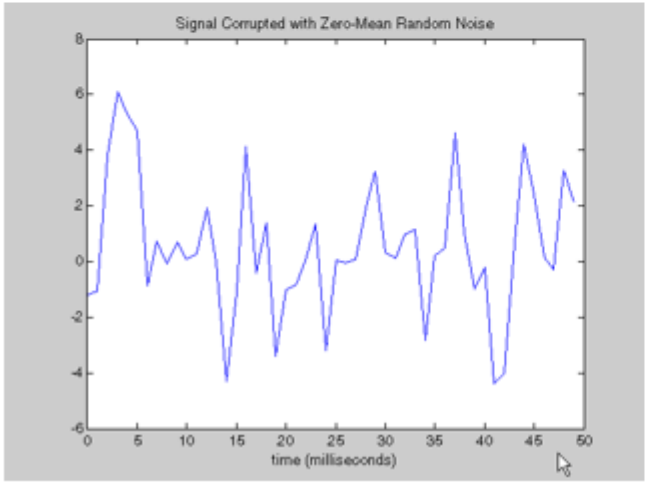
# Overall Flow Diagram: Divide-and-Conquer

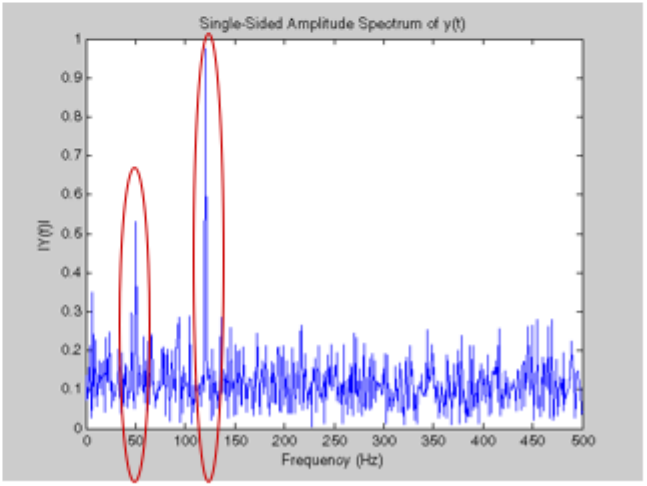


# MATLAB Implementation

X = fft(x,N) *computes the N-point DFT*

- *if the length of* $x$ *is less than* $N$, *then* $x$ *is padded with zeros.*
- *if the argument* $N$ *is omitted,*
  *then the length of the DFT is the length of* $x$.
- *if* $x$ *is a matrix,*
  *then* fft(x,N) *computes the N-point DFT of each column of* $x$.

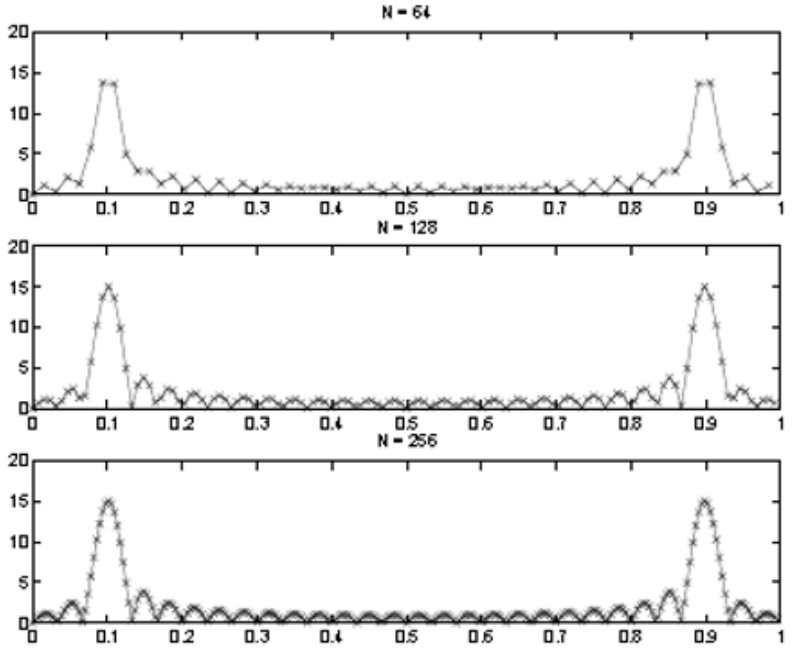


# MATLAB Implementation

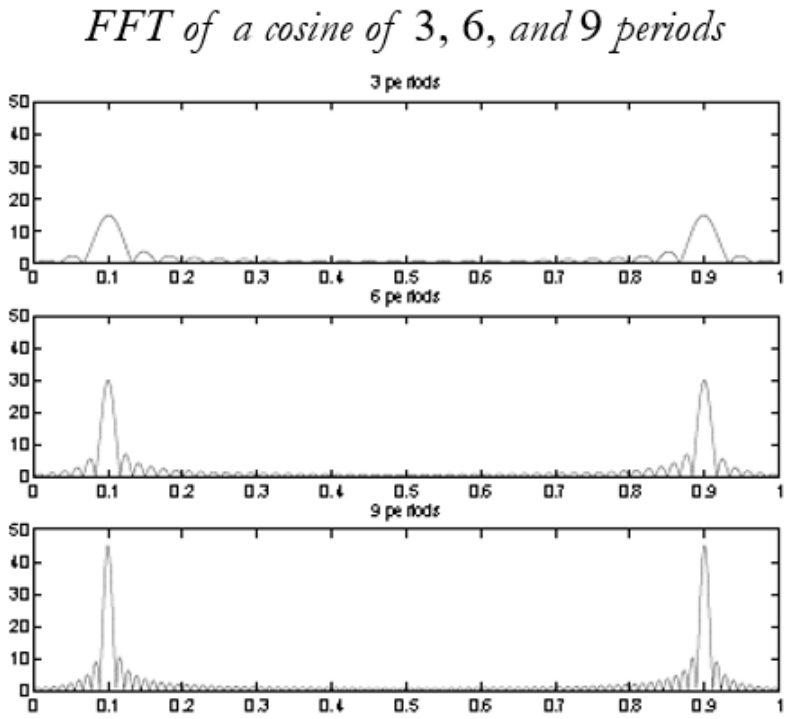
```
n = [0:29];
x = cos(2*pi*n/10);

N1 = 64;
N2 = 128;
N3 = 256;
X1 = abs(fft(x,N1));
X2 = abs(fft(x,N2));
X3 = abs(fft(x,N3));

F1 = [0 : N1 - 1]/N1;
F2 = [0 : N2 - 1]/N2;
F3 = [0 : N3 - 1]/N3;

subplot(3,1,1)
plot(F1,X1,'-x'),title('N = 64'),axis([0 1 0 20])
subplot(3,1,2)
plot(F2,X2,'-x'),title('N = 128'),axis([0 1 0 20])
subplot(3,1,3)
plot(F3,X3,'-x'),title('N = 256'),axis([0 1 0 20])
```

*FFT of a cosine for N*=64, 128, *and* 256



# MATLAB Implementation

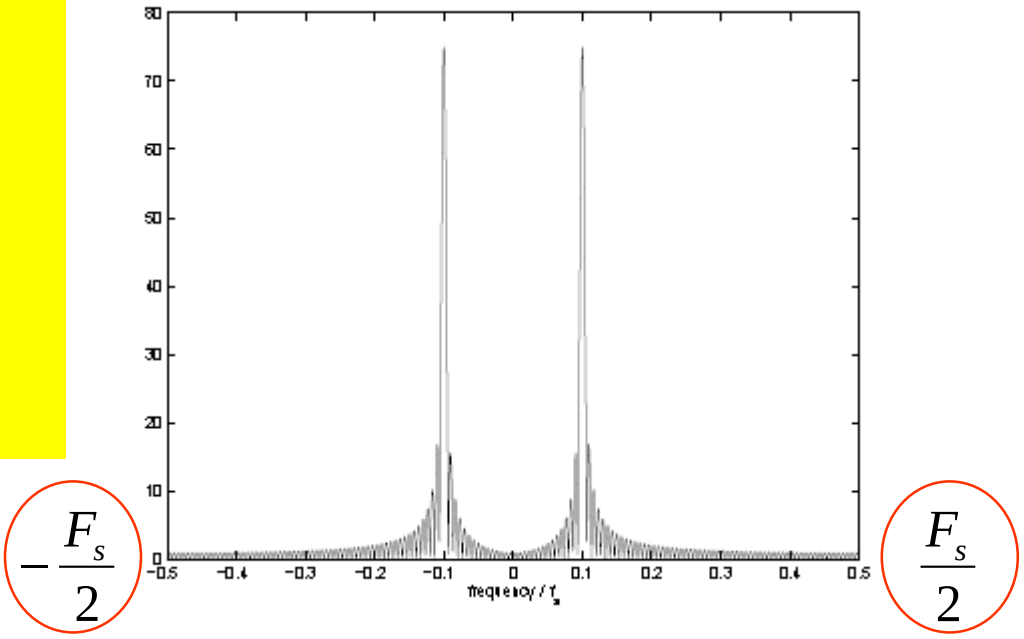
```
n = [0:29];

x1 = cos(2*pi*n/10); % 3 periods
x2 = [x1 x1]; % 6 periods
x3 = [x1 x1 x1]; % 9 periods

N = 2048;

X1 = abs(fft(x1,N));
X2 = abs(fft(x2,N));
X3 = abs(fft(x3,N));

F = [0:N-1]/N;

subplot(3,1,1)
plot(F,X1),title('3 periods'),axis([0 1 0 50])
subplot(3,1,2)
plot(F,X2),title('6 periods'),axis([0 1 0 50])
subplot(3,1,3)
plot(F,X3),title('9 periods'),axis([0 1 0 50])
```



# MATLAB Implementation

*The signal's spectrum should be entirely below* $\frac{F_s}{2}$.

```
n = [0:149];
x1 = cos(2*pi*n/10);
N = 2048;
X = abs(fft(x1,N));
X = fftshift(X);
F = [-N/2:N/2-1]/N;
plot(F,X),
xlabel('frequency / f_s')
```

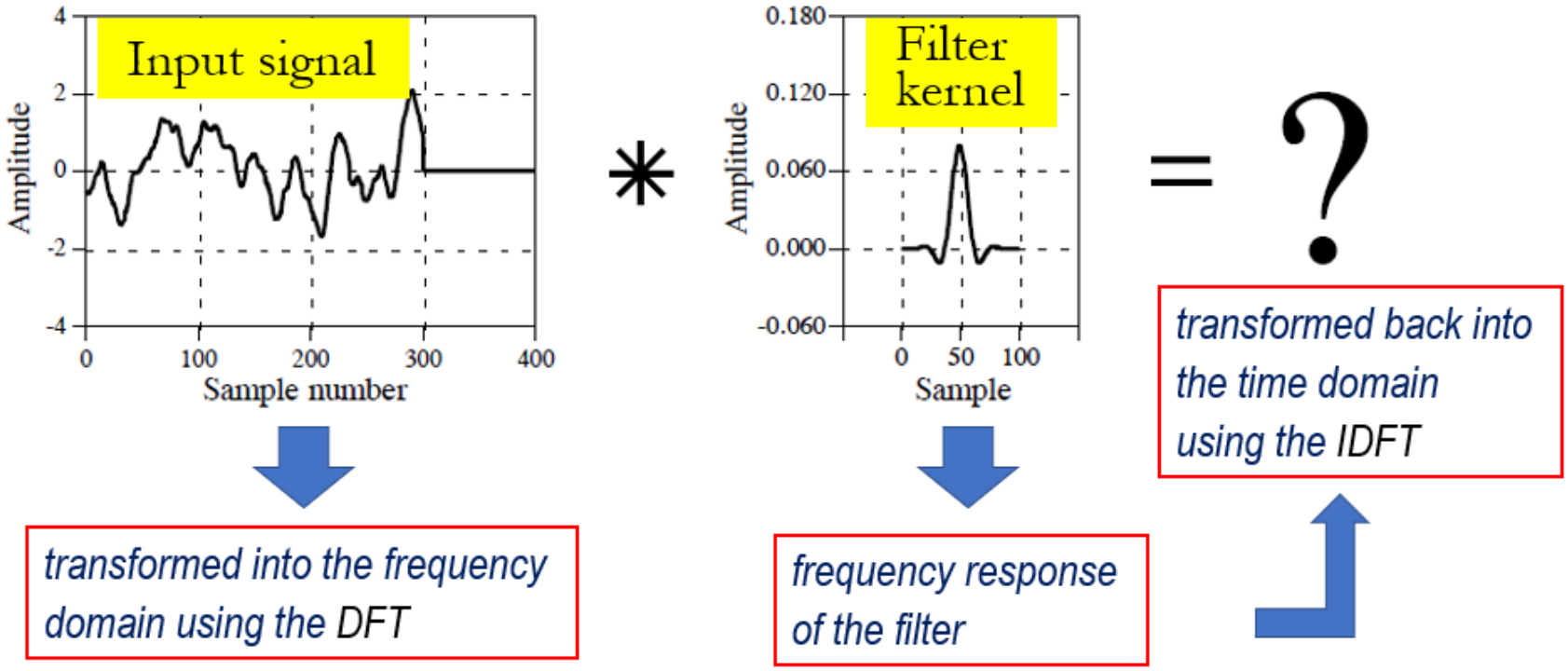



*Approximate spectrum of a sinusoid with FFT*

# FFT Convolution

# FFT Convolution (High-speed Convolution)

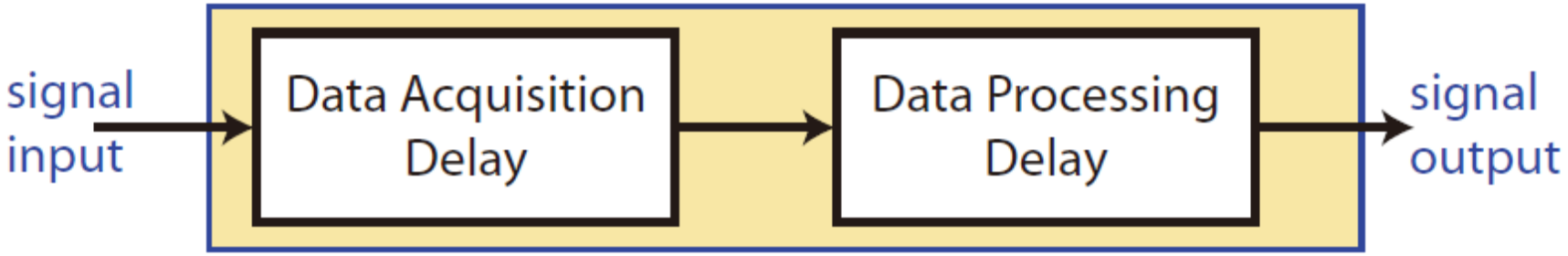
Multiplication in the frequency domain corresponds to convolution in the time domain.



# FFT Convolution (High-speed Convolution)

- **Filtering of Long Data Sequences**

*If* $N$ *is too large…*



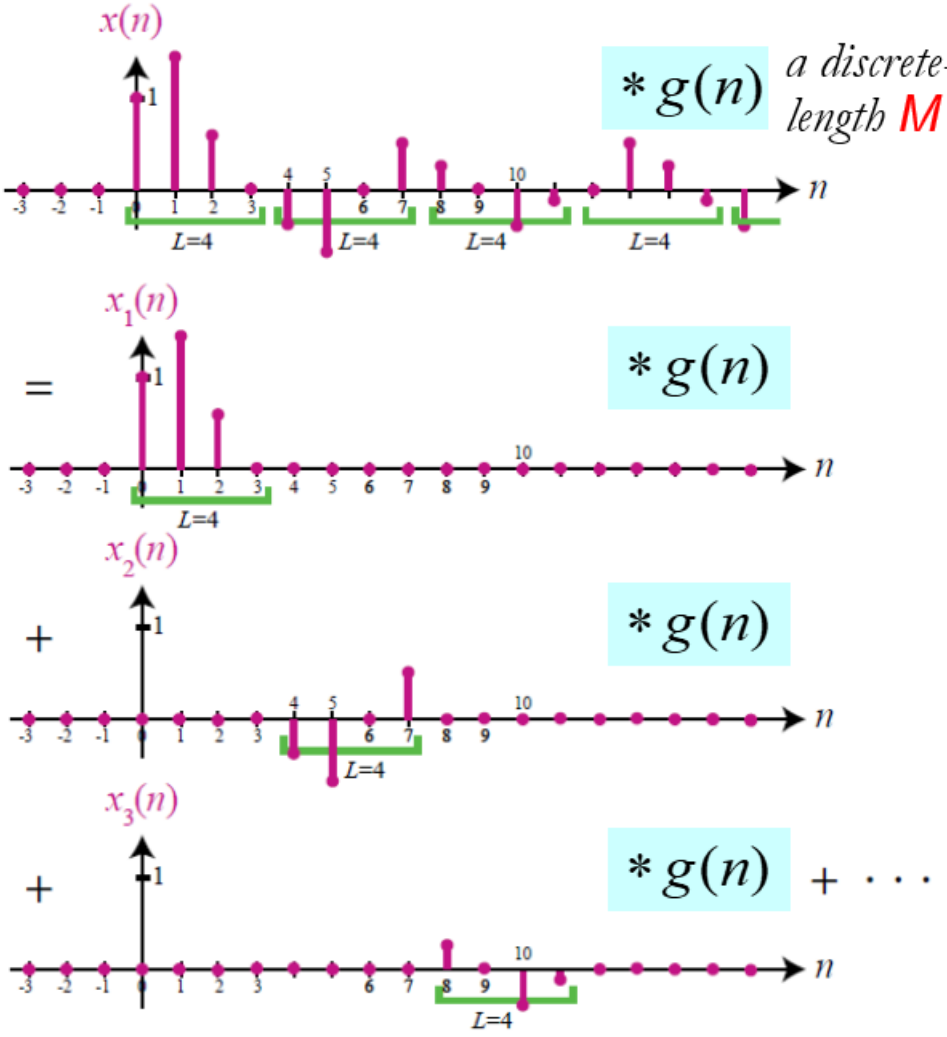

*Break into a number of segments*

Additivity:

$$\big(x_1(n)+x_2(n)\big)*g(n)=x_1(n)*g(n)+x_2(n)*g(n)$$

# FFT Convolution (High-speed Convolution)

- **Filtering of Long Data Sequences**




The linear convolution of a signal of length **L** and a signal of length **M** produces a convolved result of length

$$N = L + M - 1.$$

# Circular Convolution: Aliasing

**Example** $x_1(n) = \{1, \quad 2, \quad 2, \quad 1\}, \quad x_2(n) = \{1, \quad -1, \quad -1, \quad 1\}$

```
          1  2  2 1
          1 -1 -1 1
          1  2  2 1
       -1 -2 -2 -1
    -1 -2 -2 -1
  1  2  2 1
  1 1 -1 -2 -1 1 1
```

N=5

2 2 -1 -2 -1

*If we make both* $x_1(n)$ *and* $x_2(n)$ $N = N_1 + N_2 - 1$ *point sequences by padding an appropriate number of zeros, then the circular convolution is identical to the linear convolution.*

**Example** $x_3(n) = \{1,1,-1,-2,-1,1,1\}$

*Error Analysis*

*When* $N = \max(N_1, N_2)$ *is chosen for circular convolution, then the first* $(M\text{-}1)$ *samples are in error, where* $M = \min(N_1, N_2)$.

$$x_4(n) = x_1(n) \textcircled{6} x_2(n) = \{2,1,-1,-2,-1,1\}$$
$$e(n) = \{2,1,-1,-2,-1,1\} - \{1,1,-1,-2,-1,1\}, \quad 0 \le n \le 5$$
$$= \{1,0,0,0,0,0\}$$
$$= x_3(n+6)$$

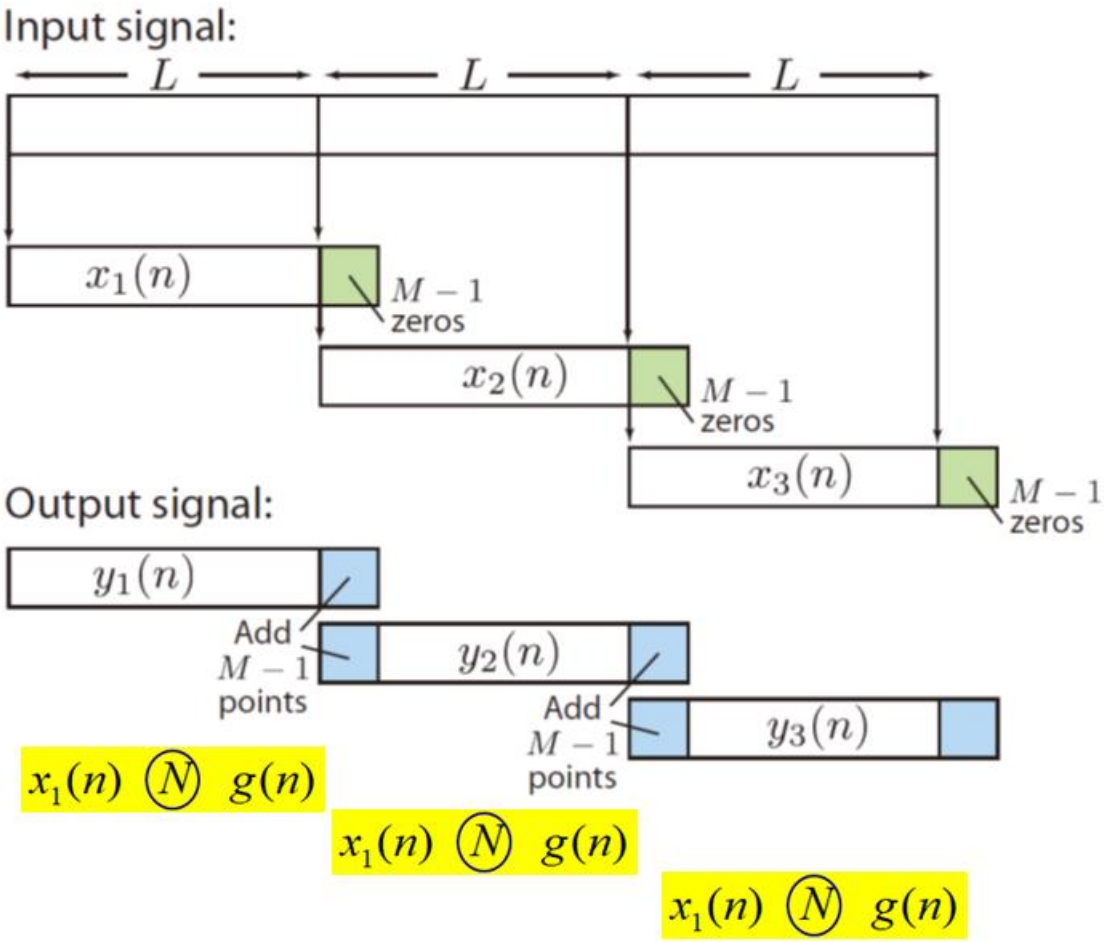
$$x_4(n) = x_1(n) \textcircled{5} x_2(n) = \{2,2,-1,-2,-1\}$$
$$e(n) = \{2,1,-1,-2,-1\} - \{1,1,-1,-2,-1\}, \quad 0 \le n \le 4$$
$$= \{1,1,0,0,0\}$$
$$= x_3(n+5)$$

$$x_4(n) = x_1(n) \textcircled{4} x_2(n) = \{0,2,0,-2\}$$
$$e(n) = \{0,2,0,-2\} - \{1,1,-1,-2\}, \quad 0 \le n \le 3$$
$$= \{-1,1,1,0\}$$
$$= x_3(n+4)$$

**Example** $x_3(n) = \{1,1,-1,-2,-1,1,1\}$

*Error Analysis*

*When* $N = \max(N_1, N_2)$ *is chosen for circular convolution, then the first* $(M\text{-}1)$ *samples are in error, where* $M = \min(N_1, N_2)$.



# Block Convolution

*We want to filter an input sequence that is being received continuously, such as a speech signal from a microphone.*

*Some practical problems:*

1. *will have to compute a large* DFT
2. *output samples are not available until all input samples are processed*
3. *an unacceptably large amount of delay*

*Therefore*

1. *segment the infinite-length input sequence into smaller sections (or blocks)*
2. *process each section using the* DFT
3. *assemble the output sequence from the outputs of each section*

**Example** $x(n)=(n+1),\quad 0\le n\le 9,\quad g(n)=\{\underset{\uparrow}{1},0,-1\}$

*the overlap-save method using* $N=6$ *to compute* $y(n)=x(n)*g(n).$

$$x_1(n)=\{0,0,1,2,3,4\}$$
$$x_2(n)=\{3,4,5,6,7,8\}$$
$$x_3(n)=\{7,8,9,10,0,0\}$$

$$N=6, M=3$$
$$N=L+M-1$$
$$L=4$$

$$y_1=x_1(n)\ \textcircled{6}\ g(n)=\{-3,-4,1,2,2,2\}$$
$$y_2=x_2(n)\ \textcircled{6}\ g(n)=\{-4,-4,2,2,2,2\}$$
$$y_3=x_3(n)\ \textcircled{6}\ g(n)=\{7,8,2,2,-9,-10\}$$

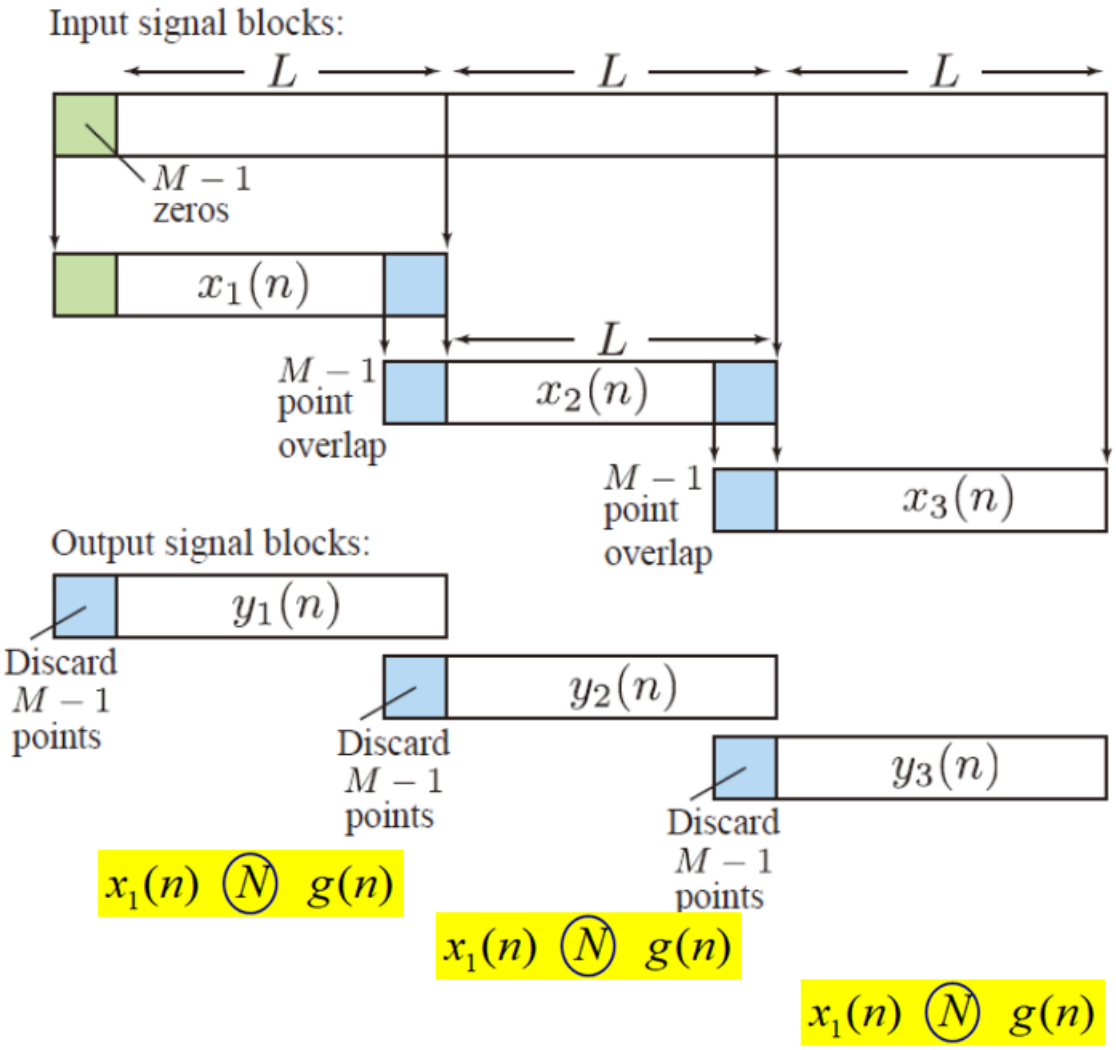
$$y(n)=\{\underset{\uparrow}{1},2,2,2,2,2,2,2,2,2,-9,-10\}$$

*Agrees with the linear convolution*

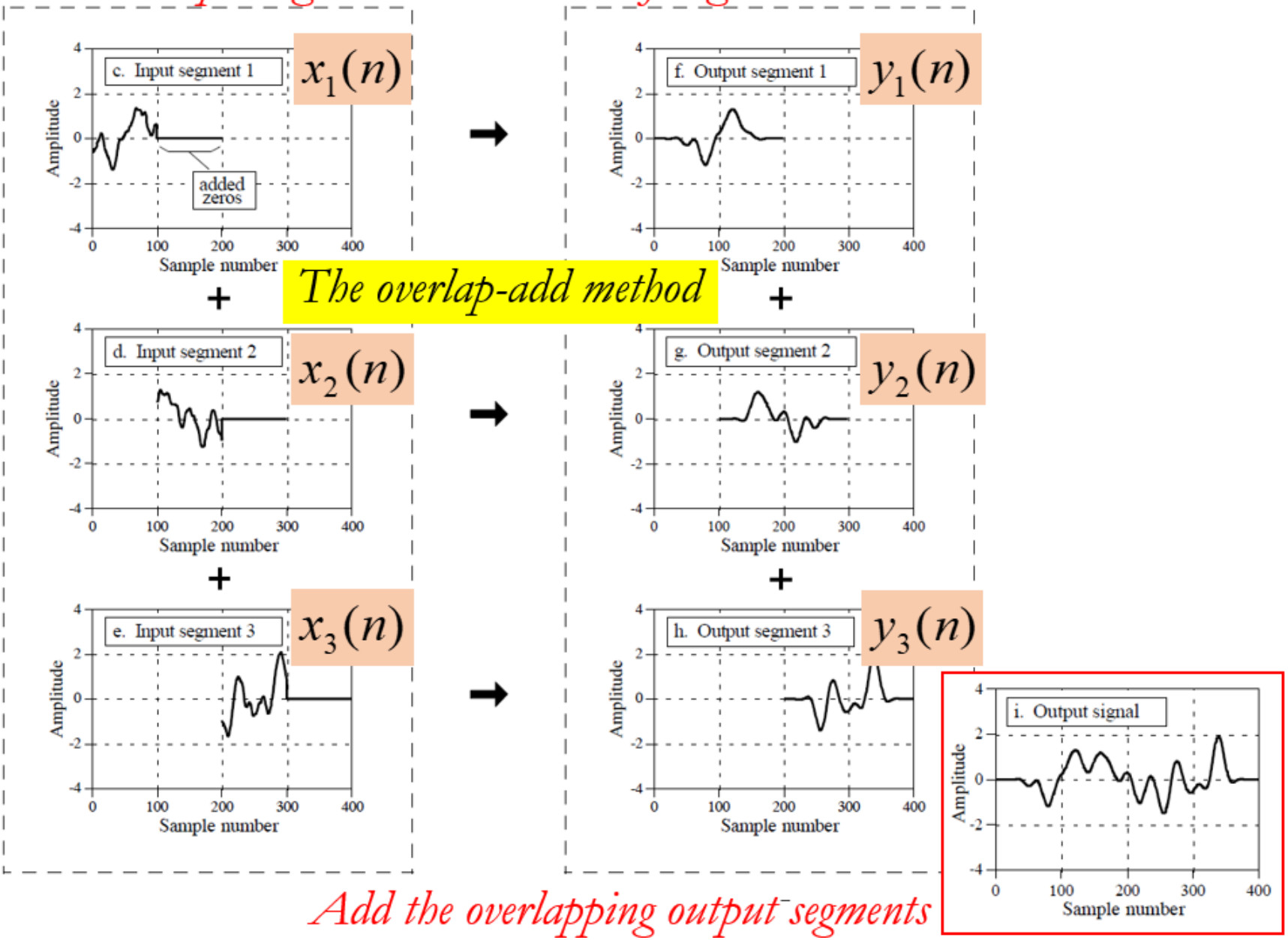
$$x(n)*g(n)=\{\underset{\uparrow}{1},2,2,2,2,2,2,2,2,2,-9,-10\}$$

**Example** $x(n) = (n+1), \quad 0 \le n \le 9, \quad g(n) = \{1, 0, -1\}$

*the overlap-save method using* $N=6$ *to compute* $y(n) = x(n) * g(n)$.



*Break the input signal into a number of segments*



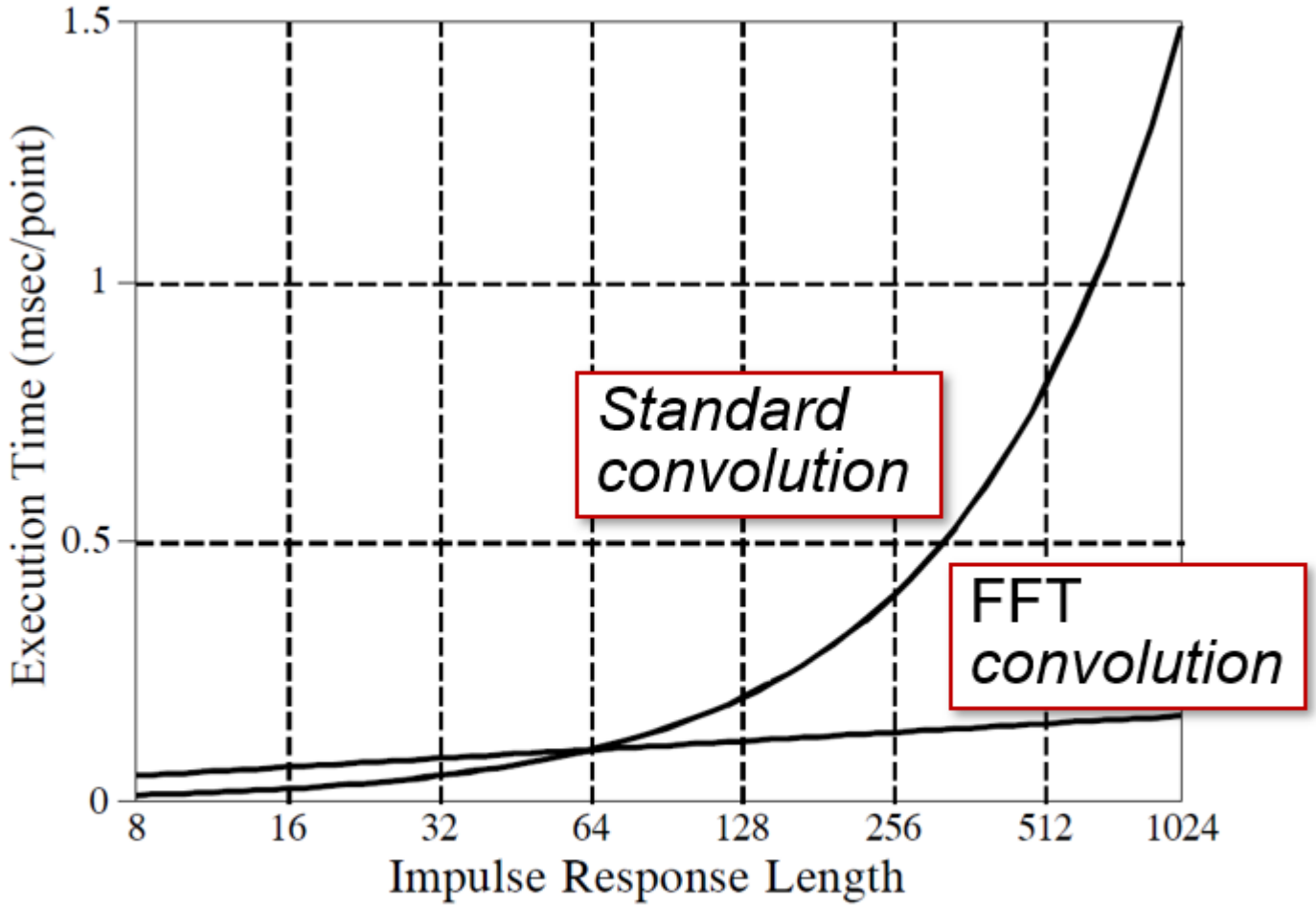

*Add the overlapping output segments*

# High-speed Block Convolution

*Replace the* DFT *by the* radix-2 FFT *algorithm*
*to obtain a high-speed overlap-and-save algorithm*

```
function [y] = hsolpsav(x,g,N)
% High-speed Overlap-Save method of block convolutions
using FFT
% ------------------------------------------------------------------------
% [y] = hsolpsav(x,g,N)
%  y = output sequence
%  x = input sequence
%  g = impulse response
%  N = block length (must be a power of two)
%
N = 2^(ceil(log10(N)/log10(2));
Lenx = length(x); M = length(g);
M1 = M-1; L = N-M1; g = fft(g,N);
%
x = [zeros(1,M1), x, zeros(1,N-1)];
K = floor((Lenx+M1-1)/(L));          % # of blocks
Y = zeros(K+1,N);
for k = 0:K
xk = fft(x(k*L+1:k*L+N));
Y(k+1,:) = real(ifft(xk.*g));
end
Y = Y(:,M:N)';  y = (Y(:))';
```

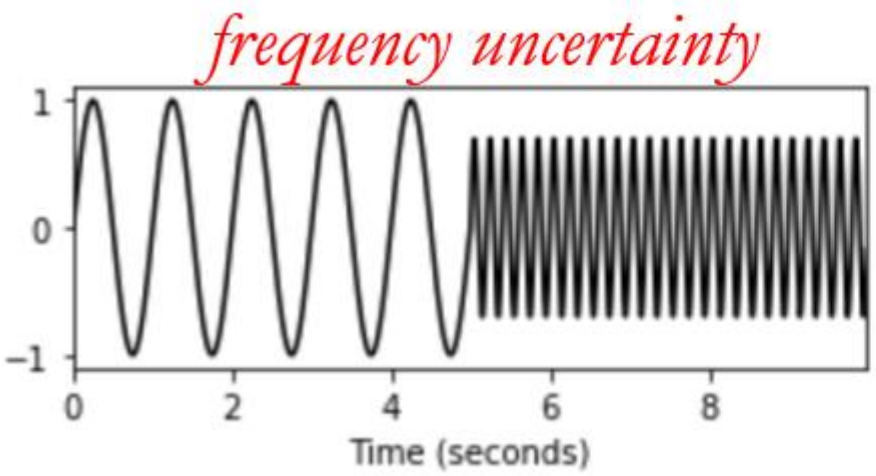
# FFT vs Standard Convolution

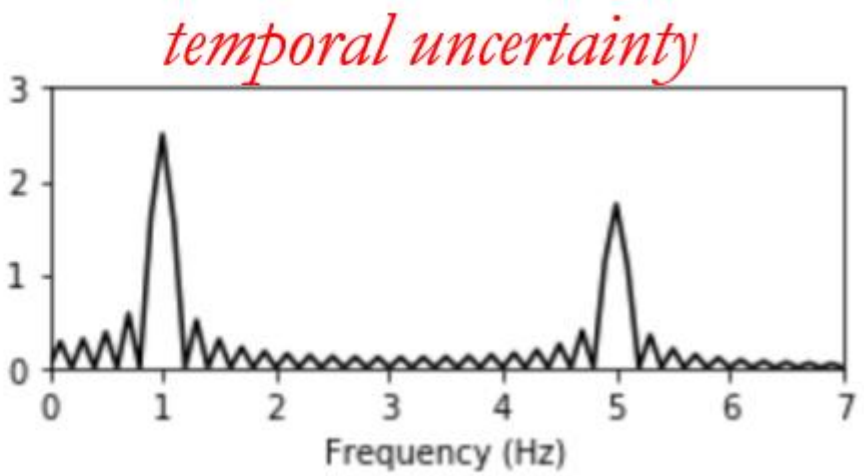



Execution times for FFT convolution

# Short-time Fourier Transform

# Short-time Fourier Transform



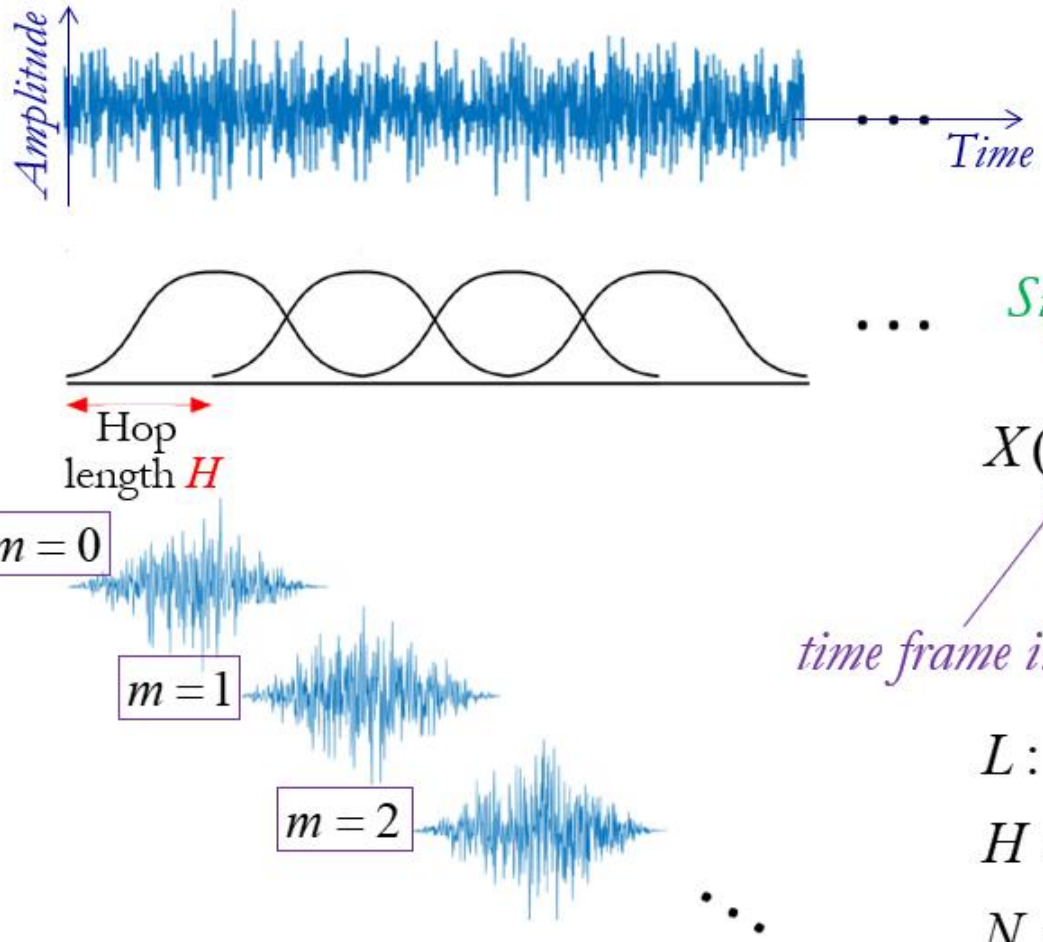

*summing across all the samples*

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

*Frequency is averaged over the entire time domain.*

*When do these frequencies occur?*

# Short-time Fourier Transform




*Signals present in the current m frame*

$$X(m,k)=\sum_{n=0}^{L-1}x(n+mH)w(n)W_N^{nk}$$

*time frame index* *frequency index*

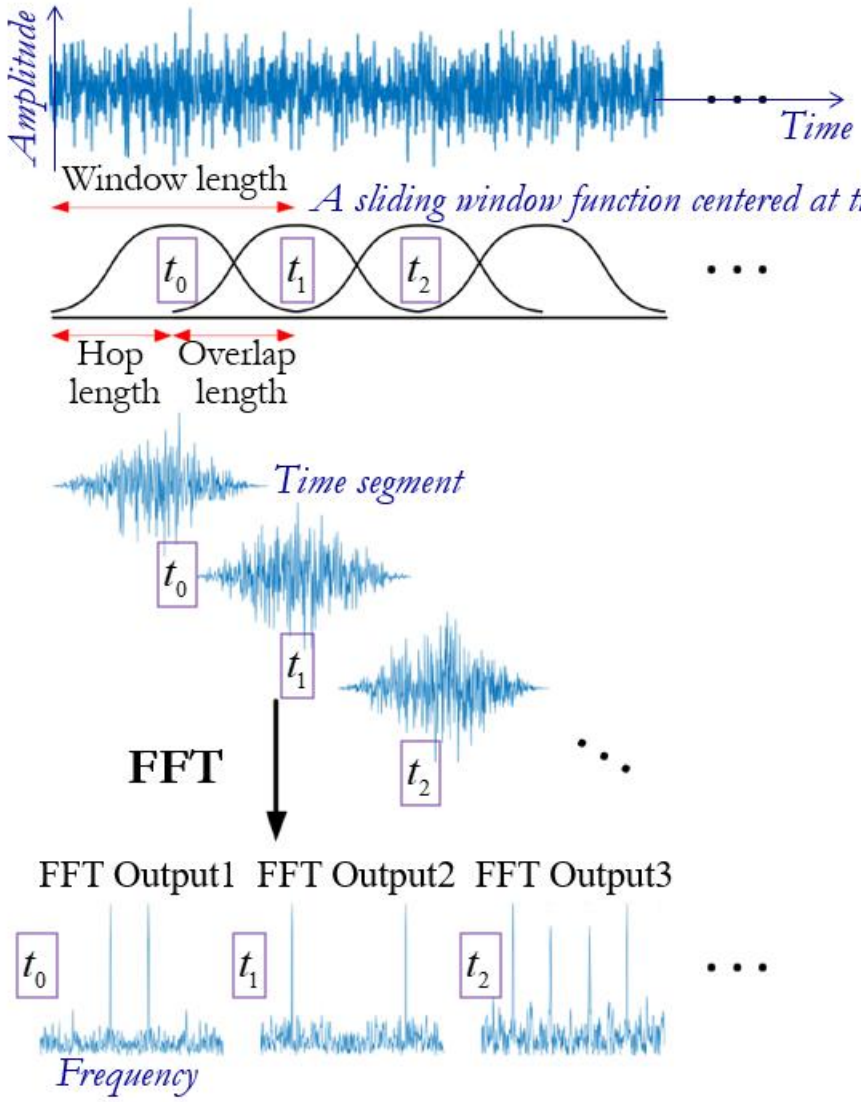
$L$ : *Window length*

$H$ : *Jump of samples*

$N$ : *DFT length*

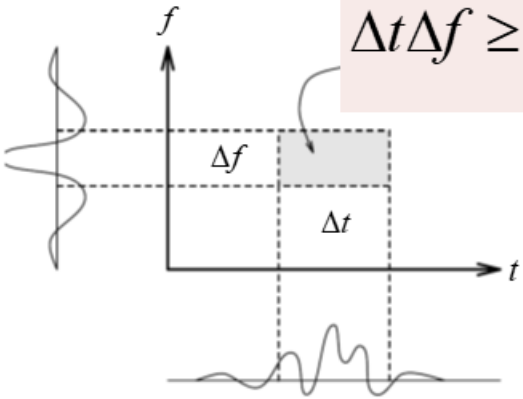
*FFT of* $x(n+mH)w(n)$

# Short-time Fourier Transform




*The product of the time and frequency resolutions*

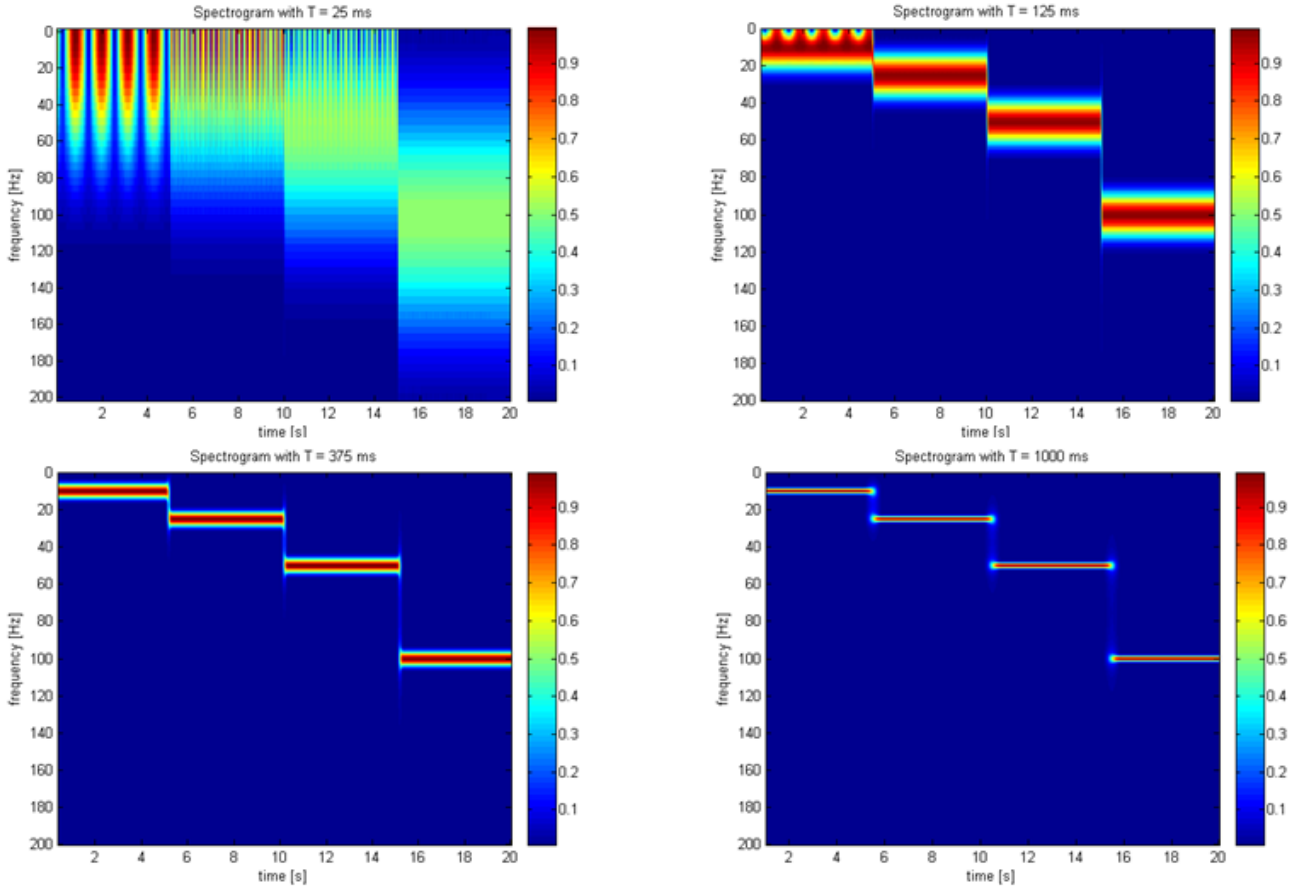
$$\Delta t \Delta f \geq \frac{1}{4\pi}$$




*Werner Karl Heisenberg* (1901-1976)

# Short-time Fourier Transform

$$x(t) = \begin{cases} \cos(2\pi 10t) & 0s \le t < 5s \\ \cos(2\pi 25t) & 5s \le t < 10s \\ \cos(2\pi 50t) & 10s \le t < 15s \\ \cos(2\pi 100t) & 15s \le t < 20s \end{cases}, \quad f_s = 400Hz$$



# Wavelet Transform

# Fourier Transform

- The Fourier Transform provides frequency information of a signal that represents frequencies and their magnitude.
- It does not tell us when in time the frequencies exist. The transform is therefore ideal for stationary signals.
- Lacks capability to provide frequency information for a localized signal region in time.

# Short-time Fourier Transform

- The STFT was developed to overcome the poor time resolution of the Fourier Transform. It gives us a time-frequency representation of the signal.
- With STFT we can assume some portion of the non-stationary signal is stationary.
- We then take a Fourier Transform of each stationary portion along the signal and add them up.

# FT and STFT

FT: $$F(\omega) = \int_{-\infty}^{+\infty} f(t) e^{-i\omega t}\, dt$$

Time localization!

STFT: $$F(\tau, \omega) = \int_{-\infty}^{+\infty} f(t)\, w(t-\tau) e^{-i\omega t}\, dt$$

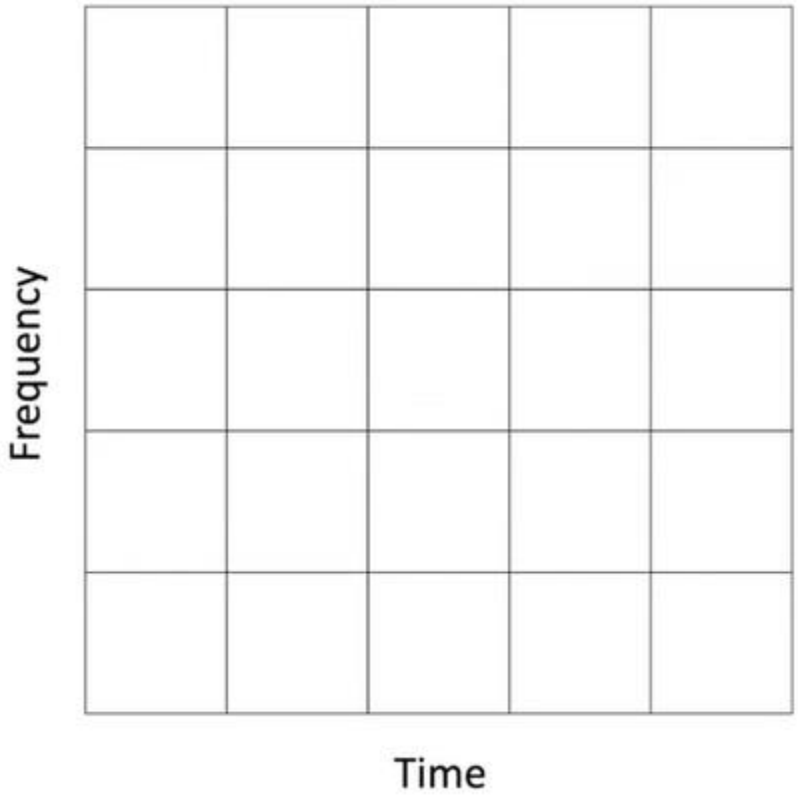
# Limitation of STFT

- The window function is finite, so frequency resolution decreases.
- Fixed length of the window means time and frequency resolutions are fixed for the entire length of the signal.

We cannot know what frequencies exist at what time instance, but we can know what frequency bands exist at what time intervals.

$$\Delta t \Delta f \geq \frac{1}{4\pi}$$

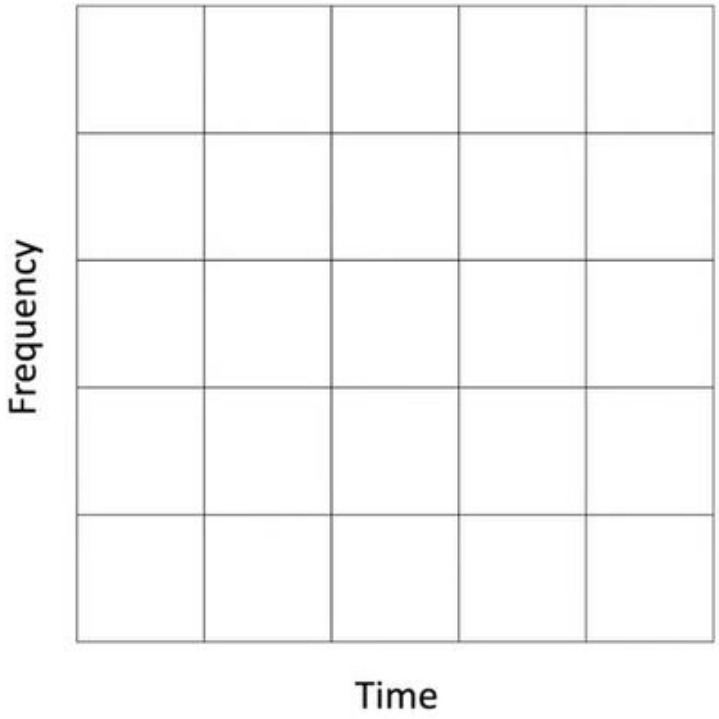
# Limitation of STFT




Narrow Window = Good time resolution, poor frequency resolution

Wide Window = Poor time resolution, good frequency resolution

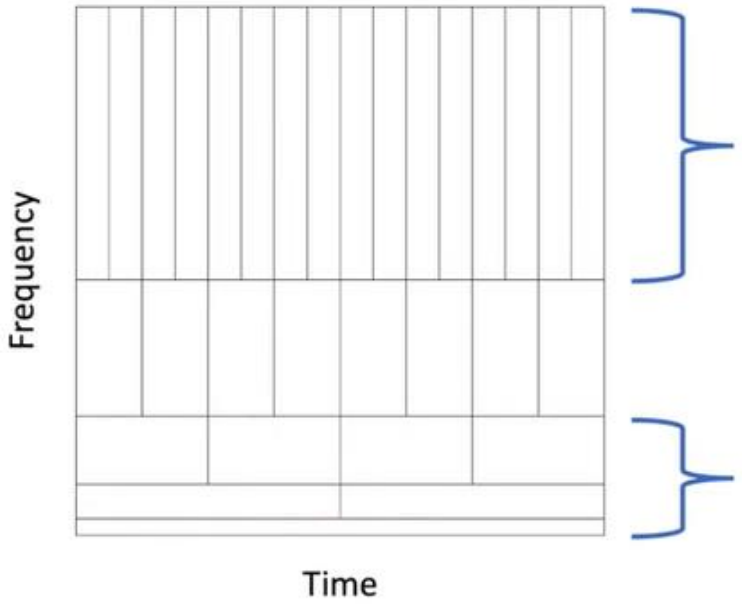
# Limitation of STFT




- Low frequency components often last a long period of time, so a high frequency resolution is required.
- High frequency components often appear as short bursts, invoking the need for a higher time resolution.

# Wavelet Transform

The Wavelet Transform results in analyzing a signal into different frequencies at different resolutions, known as multiresolution analysis.




Good time resolution and poor frequency resolution at high frequencies.

Good frequency resolution and poor time resolution at low frequencies.

# Wavelet Transform

Obtained from the function $\psi(t)$, known as the wavelet.

$$F(\tau, s) = \frac{1}{\sqrt{|s|}} \int_{-\infty}^{+\infty} f(t)\, \psi^*\left(\frac{t-\tau}{s}\right) dt$$

Scale Parameter (1/frequency)

The wavelet is a small wave such as these…

The Wavelet is our new basis function, and acts as a window function.

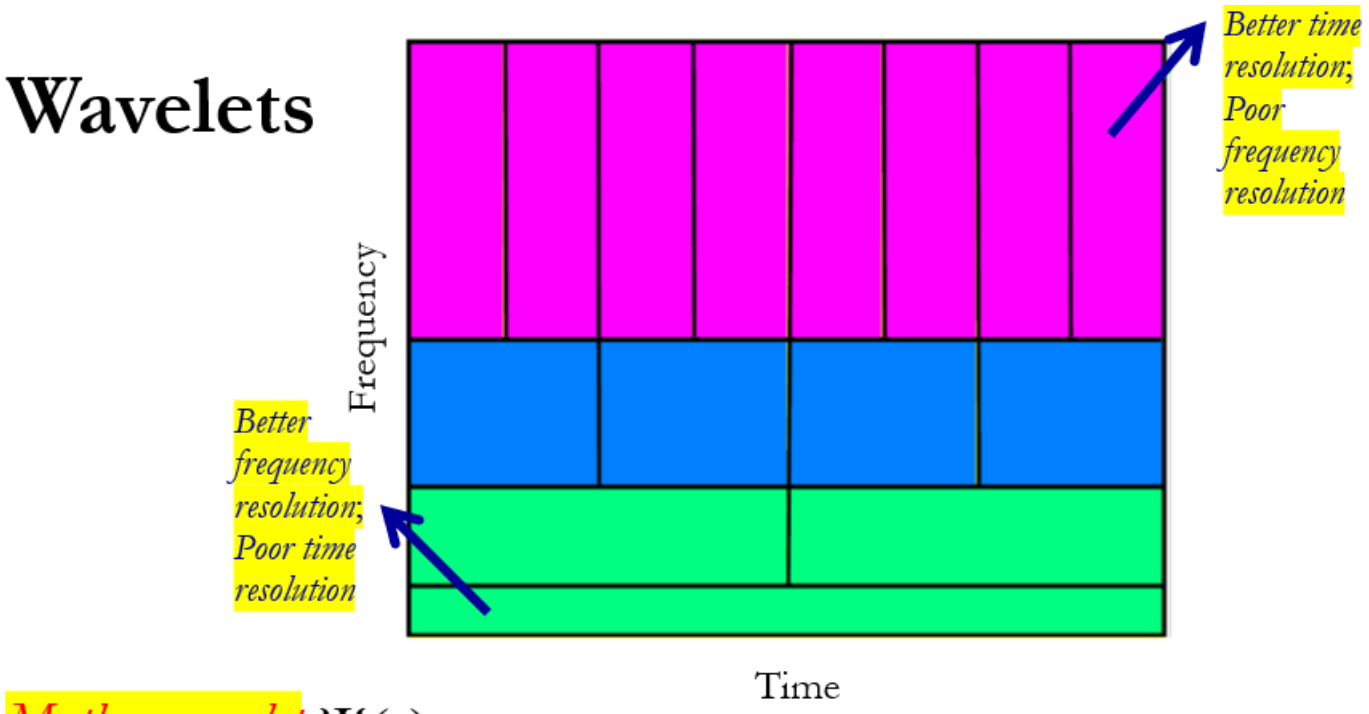
# Wavelet Transform

We can change the width of the wavelet and its central frequency as we move it across our signal by changing s. This is called scaling.

Expanded wavelet is better at resolving low frequency components of the signal with bad time resolution.
(Large values of s)

Shrunken wavelet is better at resolving high frequency components of the signal with good time resolution.
(Small values of s)

# Mother Wavelet




*Mother wavelet* $\Psi(t)$

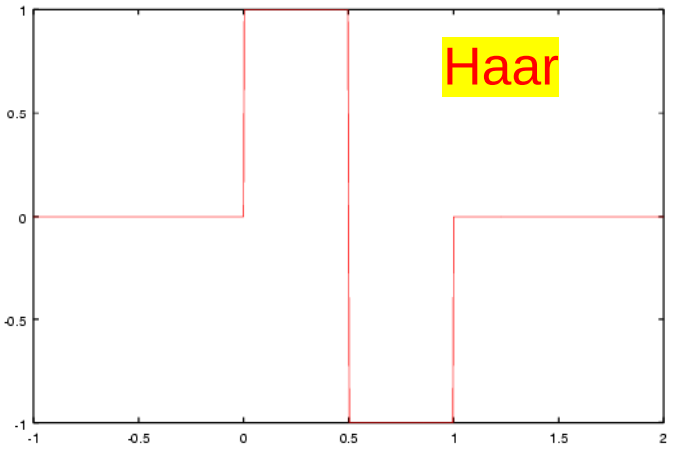
$$W_\Psi f(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t) \Psi^* \left( \frac{t-b}{a} \right) dt$$

$$\bar{\Psi}_{a,b}(t) = \frac{1}{\sqrt{a}} \Psi \left( \frac{t-b}{a} \right)$$

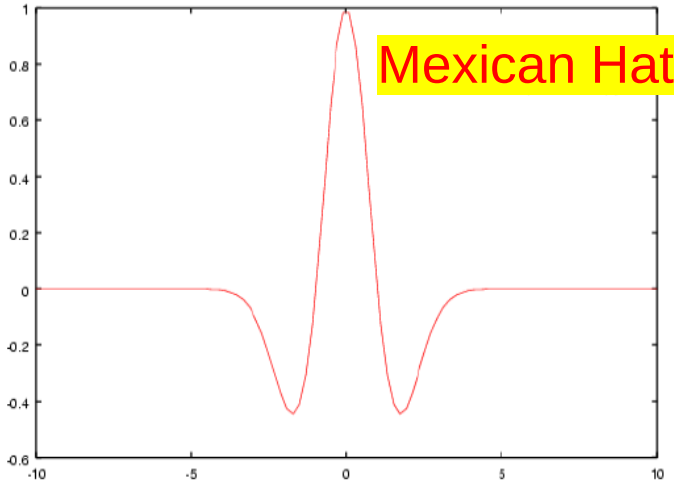
$a$ : *scale*, $b$ : *translation*

# Example

$$\Psi(t) = \begin{cases} 1 & 0 \le t < 0.5 \\ -1 & 0.5 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$

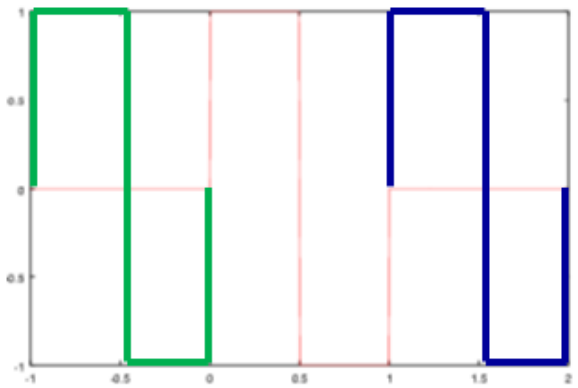



$$\Psi(t) = \left(1 - t^2\right) e^{-\frac{t^2}{2}}$$



# Orthogonal Wavelets - Dyadic Haar

$$\bar{\Psi}_{i,n}(t) = \frac{1}{\sqrt{2^i}} \Psi\left( \frac{t - 2^i n}{2^i} \right), \quad i = [0,1,2,\cdots]$$



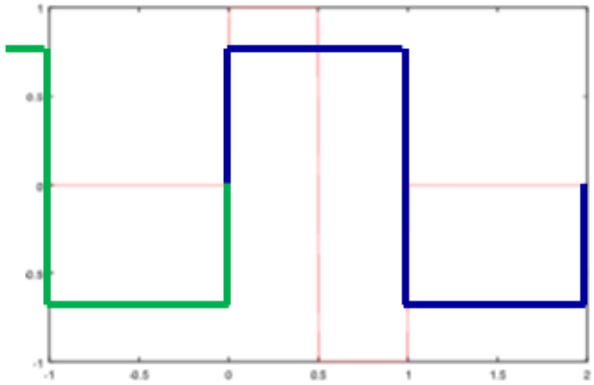
$$\bar{\Psi}_{0,0}(t) = \frac{1}{\sqrt{2^0}} \Psi\left( \frac{t - 2^0 0}{2^0} \right) = \Psi(t)$$

$$\bar{\Psi}_{0,1}(t) = \frac{1}{\sqrt{2^0}} \Psi\left( \frac{t - 2^0 1}{2^0} \right) = \Psi(t-1)$$

$$\bar{\Psi}_{0,-1}(t) = \frac{1}{\sqrt{2^0}} \Psi\left( \frac{t + 2^0 1}{2^0} \right) = \Psi(t+1)$$

# Orthogonal Wavelets - Dyadic Haar

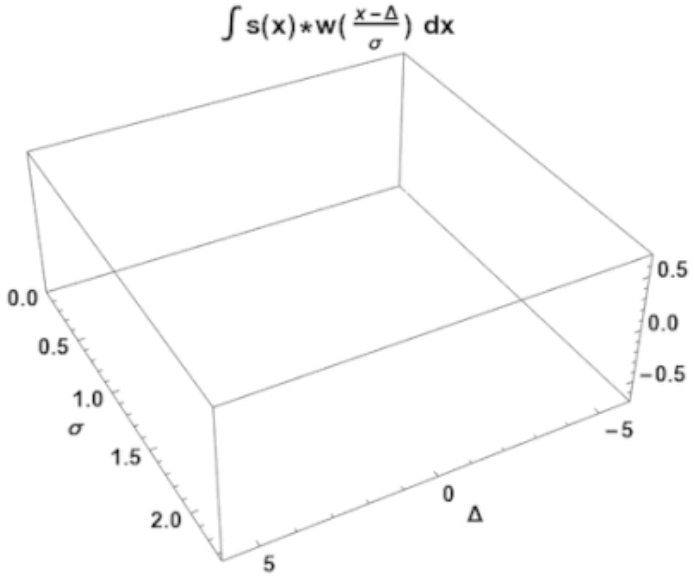
$$\bar{\Psi}_{i,n}(t) = \frac{1}{\sqrt{2^i}} \Psi\left( \frac{t - 2^i n}{2^i} \right), \quad i = [0,1,2,\cdots]$$



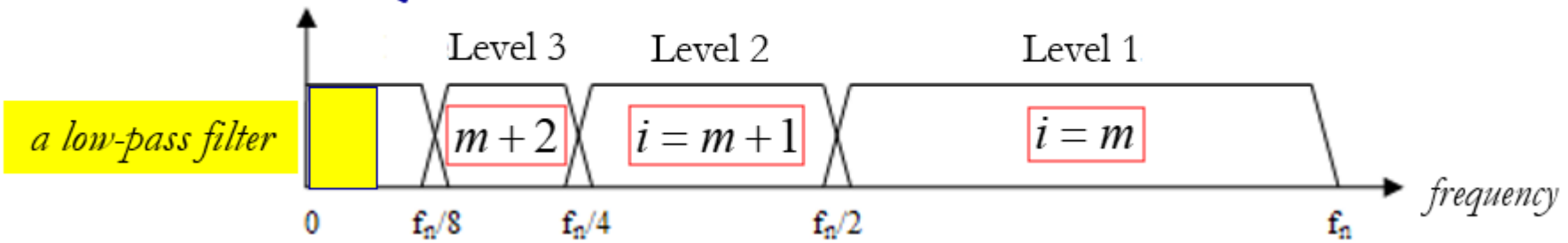
$$\bar{\Psi}_{1,0}(t) = \frac{1}{\sqrt{2^1}} \Psi\left( \frac{t - 2^1 0}{2^1} \right) = \frac{1}{\sqrt{2}} \Psi\left( \frac{t}{2} \right)$$

$$\bar{\Psi}_{1,-1}(t) = \frac{1}{\sqrt{2^1}} \Psi\left( \frac{t + 2^1 1}{2^1} \right) = \frac{1}{\sqrt{2}} \Psi\left( \frac{t+2}{2} \right)$$

# Orthogonal Wavelets - Dyadic Haar



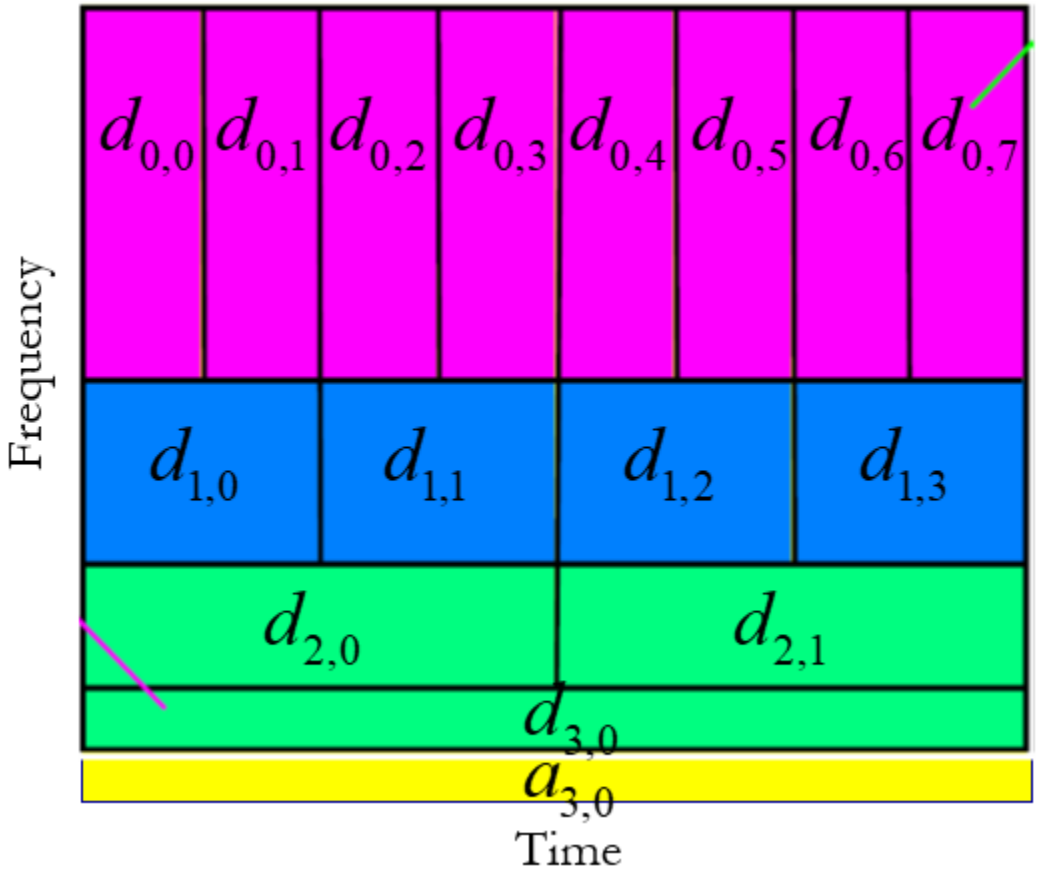

*Stretching by a factor of 2: a lower frequency more concentrated (less bandwidth)*



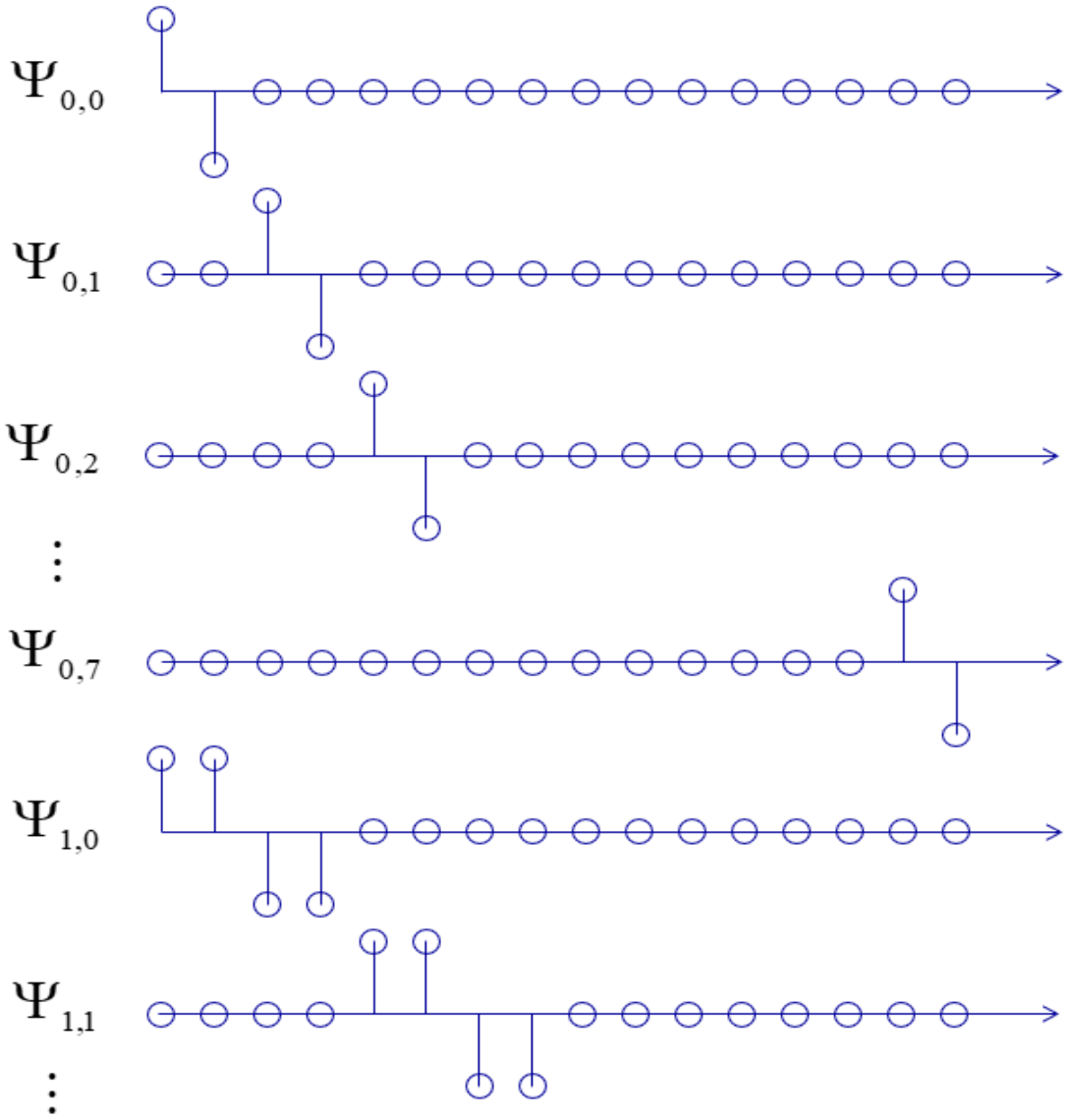
# Discrete Wavelets Transform

$$d_{a,b} = \sum_{n=0}^{N-1} x[n]\Psi_{a,b}[n]$$

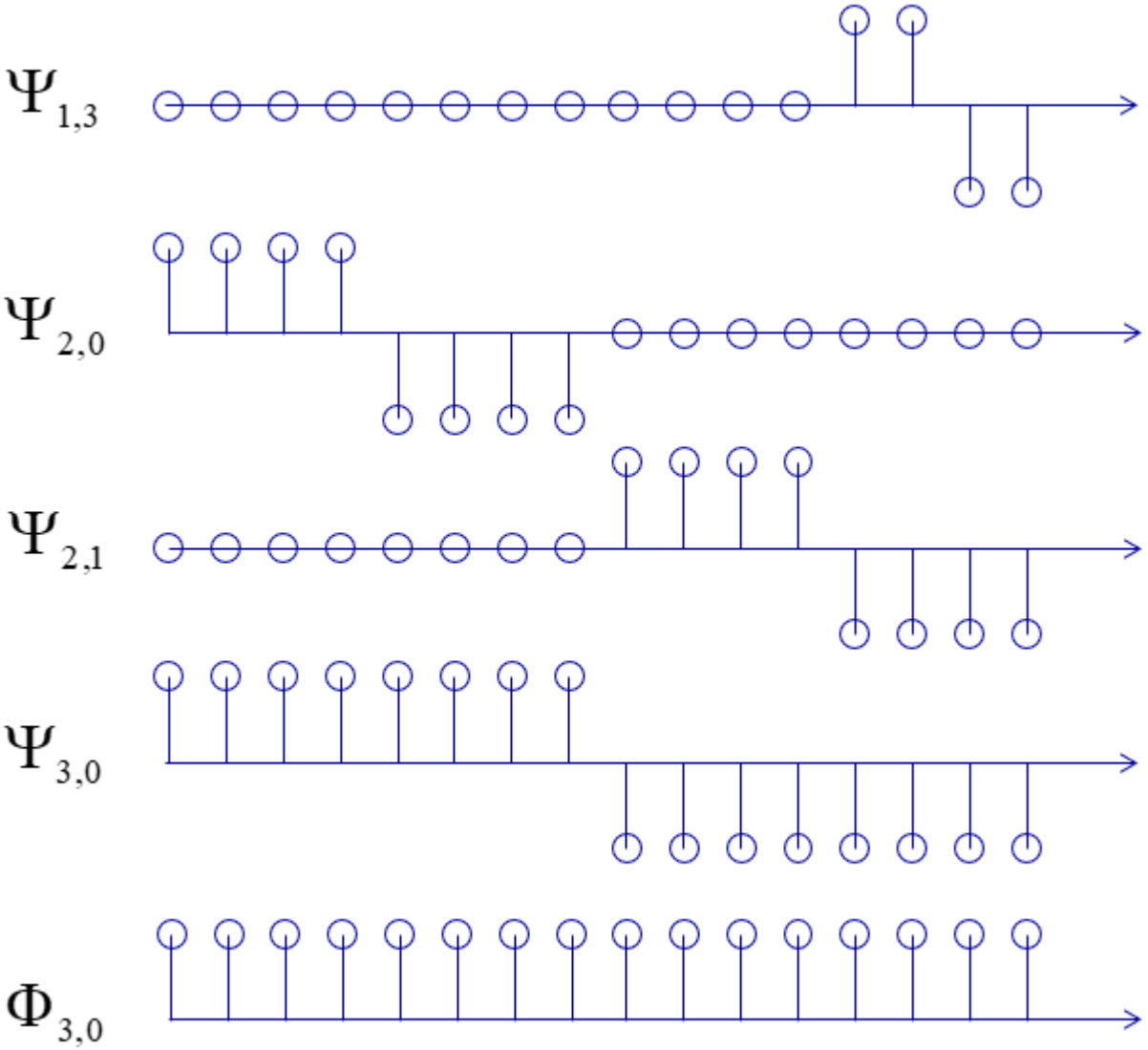
$$a_{a,b} = \sum_{n=0}^{N-1} x[n]\Phi_{a,b}[n]$$



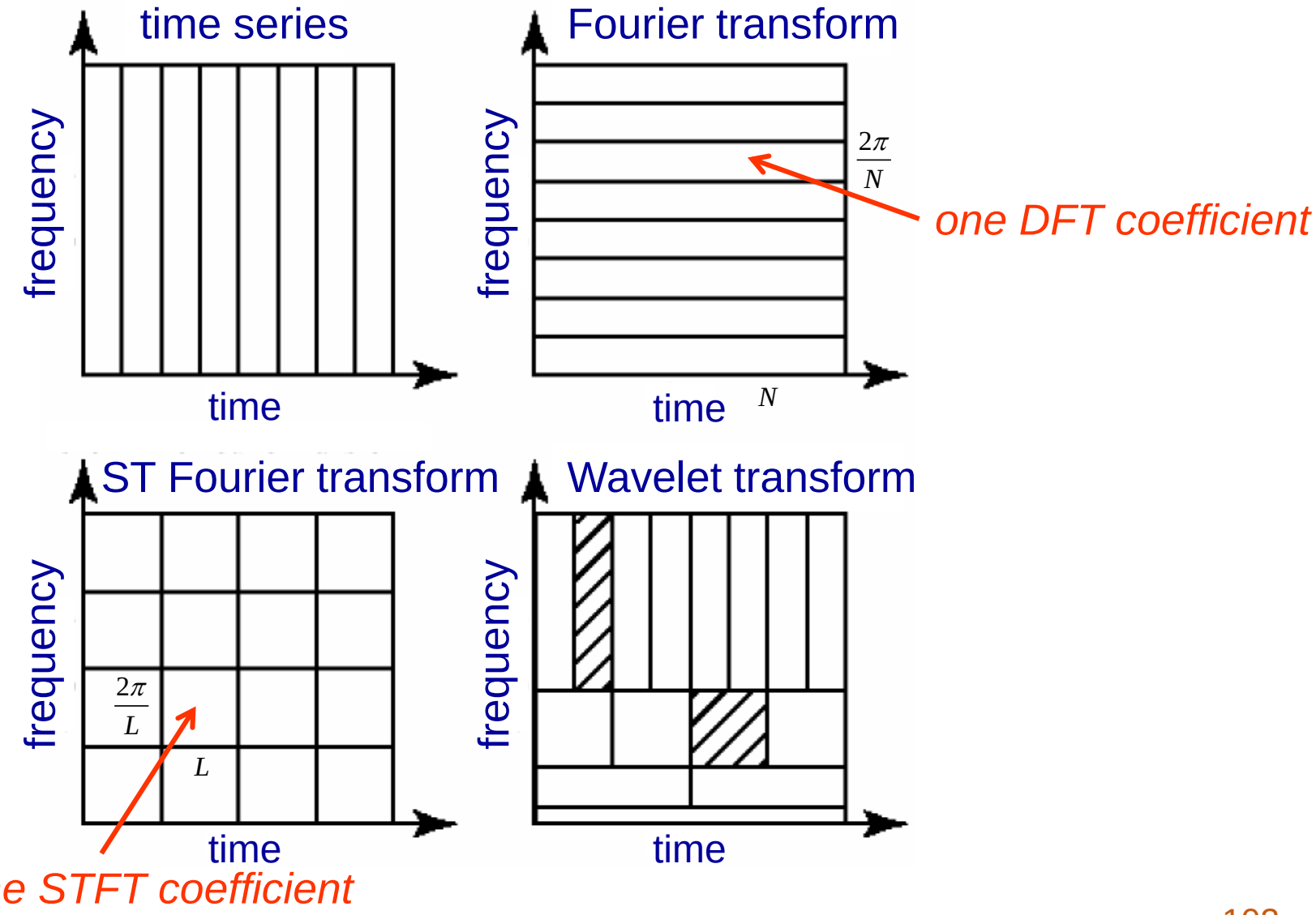
# Discrete Wavelets Transform



# Discrete Wavelets Transform



# Summary


time series
frequency
time
Fourier transform
frequency
$\frac{2\pi}{N}$
time
$N$
one DFT coefficient
ST Fourier transform
frequency
$\frac{2\pi}{L}$
$L$
time
one STFT coefficient
Wavelet transform
frequency
time

# Thank you


JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY
1990