第四章 快速傅里叶变换

Fast Forurier Transform

第四章　快速傅里叶变换

Fast Forurier Transform

4.2 基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法原理 (1)

华东理工大学信息科学与工程学院 万永菁

4.2 基于时间抽取的基-2-FFT快速算法

一、算法基本思想

设输入序列长度为$N=2^M$(M为正整数)，将该序列按时间顺序的奇偶分解为越来越短的子序列，称为基2按时间抽取的FFT算法。也称为Coolkey-Tukey算法。

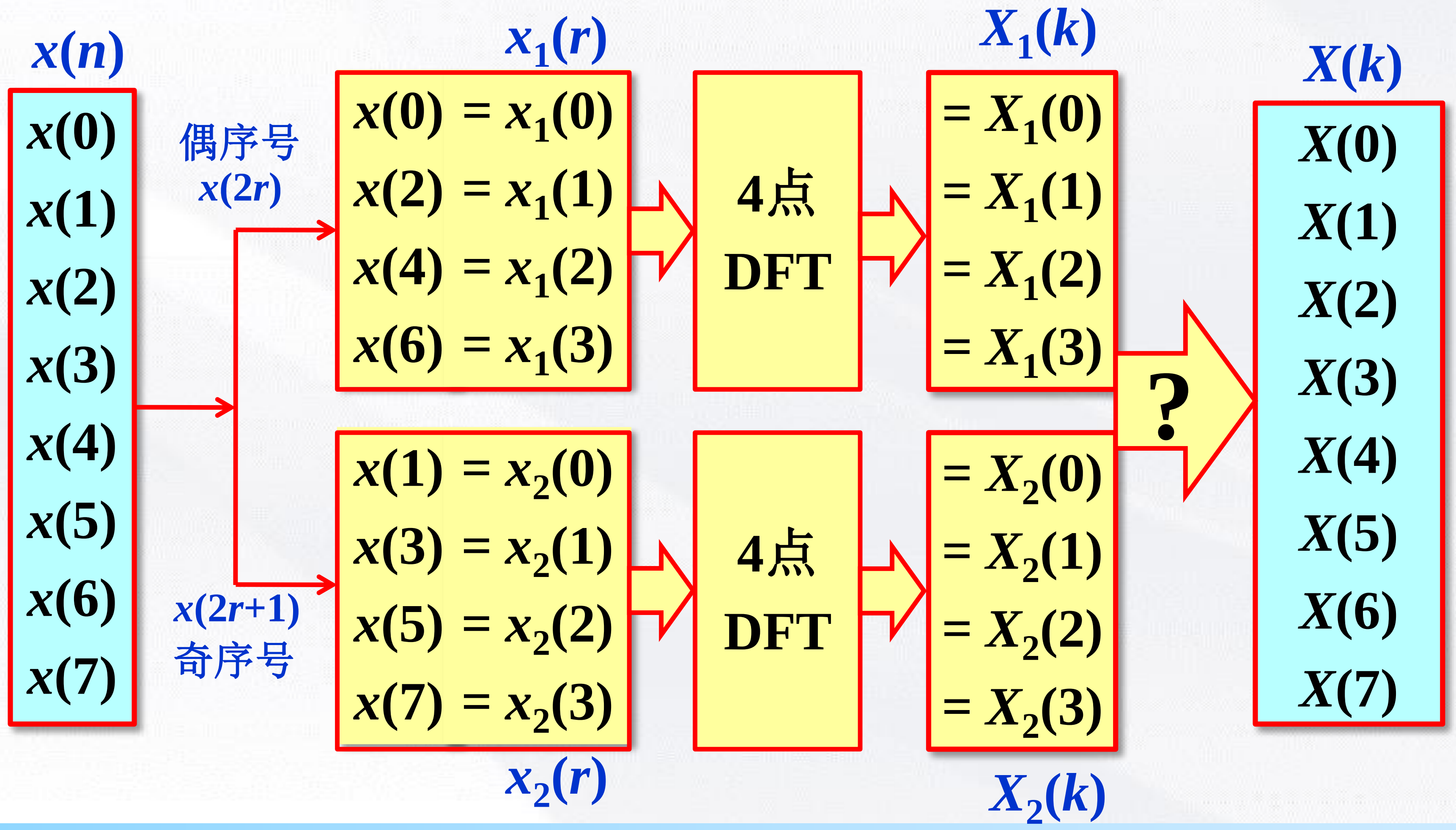
其中基2表示：$N=2^M$，M为整数.若不满足这个条件，可以人为地加上若干零值（加零补长）使其达到 $N=2^M$。

4.2 基于时间抽取的基-2-FFT快速算法

二、算法原理与分析

- 前提：保证$x(n)$的长度$N=2^M$，可通过序列后添零来实现。

设$N=8$，则$M=3$



➢ 基于时间奇偶序号分解

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

$$\boxed{X(k)} = \mathrm{DFT}[x(n)] = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$\boxed{k \in [0, N-1]}$$

$$= \sum_{r=0}^{N/2-1} \underbrace{x(2r)}_{\text{偶序号}} W_N^{2rk} + \sum_{r=0}^{N/2-1} \underbrace{x(2r+1)}_{\text{奇序号}} W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{N/2-1} x_1(r) \boxed{W_N^{2rk}} + W_N^{k} \sum_{r=0}^{N/2-1} x_2(r) \boxed{W_N^{2rk}}$$

$$= \sum_{r=0}^{N/2-1} x_1(r) \boxed{W_{N/2}^{rk}} + W_N^{k} \sum_{r=0}^{N/2-1} x_2(r) \boxed{W_{N/2}^{rk}}$$

$$\boxed{= X_1(k) + W_N^{k} X_2(k)}$$

$$\boxed{k \in [0, N/2 - 1]}$$

$$\boxed{k \in [0, N/2 - 1]}$$

只能求出前一半$X(k)$！

华東理工大學

➤ 已知 $X(k)$ 前一半的值，如何求出其后一半的值?

$$X(k)=X_1(k)+W_N^k X_2(k)$$

$$k\in[0,N/2-1]$$

$$k=k+N/2$$

$$(k+N/2)\in[N/2,N-1]$$

$$X(k+N/2)=X_1(k+N/2)+W_N^{k+N/2}X_2(k+N/2)$$

$$X(k)=X_1(k)+W_N^k X_2(k)$$

$k\in[0,N/2-1]$

$k=k+N/2$

$(k+N/2)\in[N/2,N-1]$

$$X(k+N/2)=X_1(k+N/2)+W_N^{k+N/2}X_2(k+N/2)$$

DFT隐含着周期性

$$X_1(k)=\mathrm{DFT}[x_1(r)]=X_1((k))_{N/2}R_{N/2}(k)$$

$$X_1(k)=X_1(k+N/2) \qquad X_2(k)=X_2(k+N/2)$$

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

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

4.2 基于时间抽取的基-2-FFT快速算法

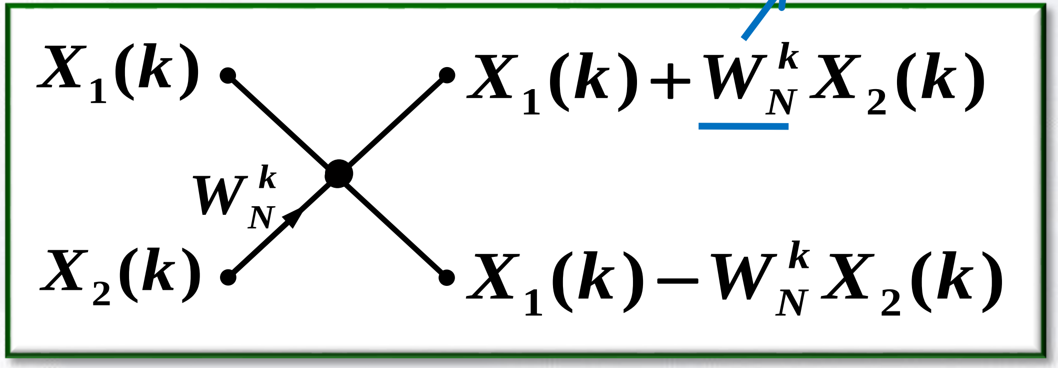
$$\underbrace{X(k)}_{\text{前半部分}} = X_1(k) + W_N^k X_2(k)$$

$$k = 0,1,\cdots,{}^{N}\!/_{2}-1$$

$$\underbrace{X(N/2+k)}_{\text{后半部分}} = X_1(k) - W_N^k X_2(k)$$

蝶形运算流图符号：

Twiddle factor
(旋转因子)



1个蝶形运算需要<u>1次复乘</u>，<u>2次复加</u>

Butterfly computation

➢ 经过一次奇偶分解后，**运算效率**是否提升?

$$X_1(k) = \mathrm{DFT}[x_1(n)]$$

N/2点DFT

$$X_2(k) = \mathrm{DFT}[x_2(n)]$$

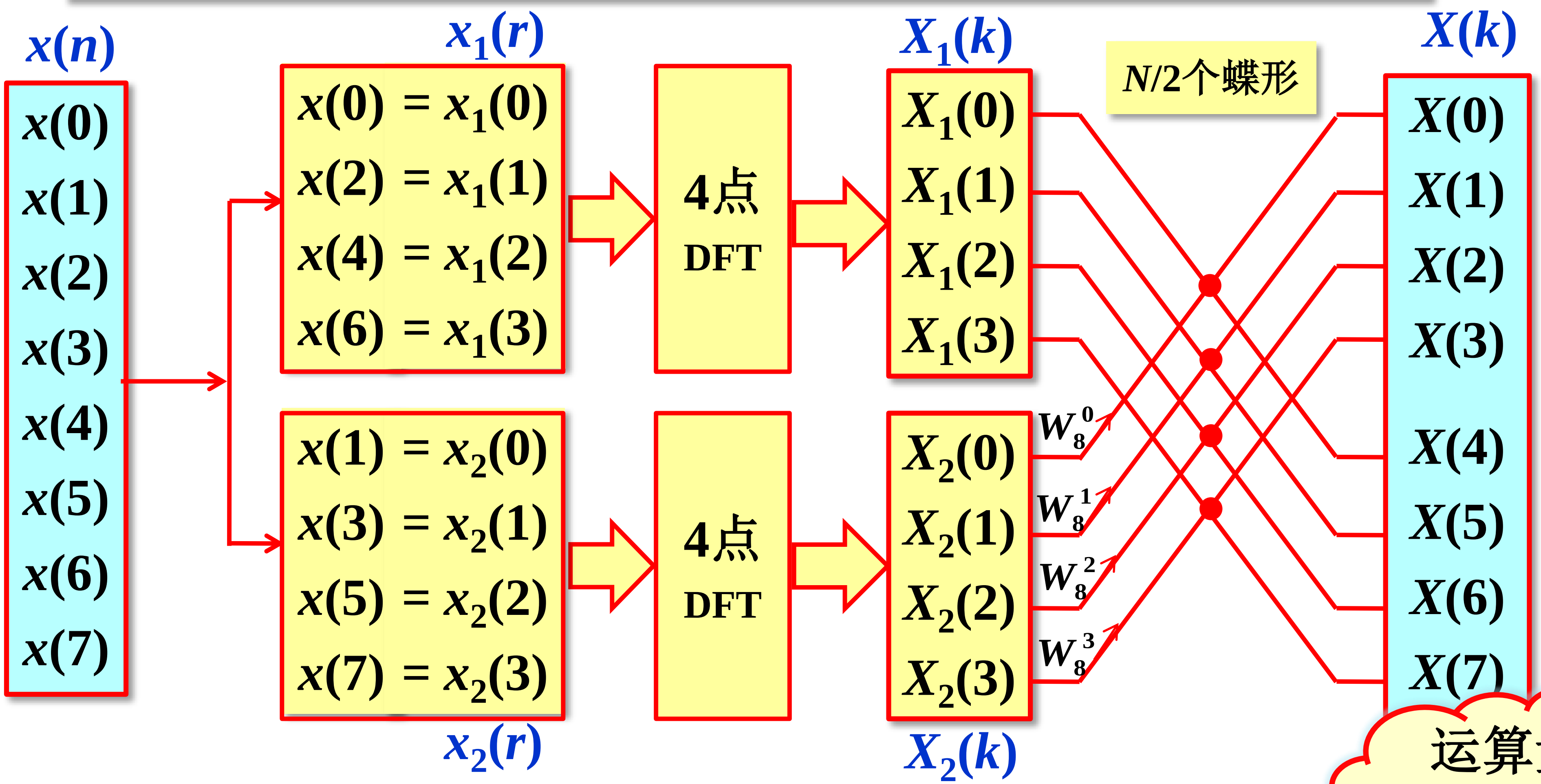
N/2点DFT

$$\begin{cases} X(k) = X_1(k) + W_N^k X_2(k) \\ X(N/2+k) = X_1(k) - W_N^k X_2(k) \end{cases}$$

N/2个蝶形运算

$$k \in [0, N/2 - 1]$$

N点DFT的一次时域抽取分解图及运算量(N=8)



运算量
明显减少！

- 直接计算： N^2=64次复数乘法
- 一次抽取： $\left(\frac{N}{2}\right)^2+\left(\frac{N}{2}\right)^2+\frac{N}{2}$ = 4^2+ 4^2+4 = 36次复数乘法

第四章　快速傅里叶变换

Fast Forurier Transform

4.2 基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法原理 (1)

华东理工大学信息科学与工程学院 万永菁

第四章　快速傅里叶变换

Fast Forurier Transform

4.2 基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法原理（2）

华东理工大学信息科学与工程学院 万永菁

4.2 基于时间抽取的基-2-FFT快速算法

因为4点DFT还是比较麻烦，所以再**继续分解**。

若将$N/2$(4点)子序列按奇/偶分解成两个$N/4$点(2点)子序列。即对将$x_1(r)$和$x_2(r)$**分解成奇、偶两个$N/4$点**，即2点的子序列。

$$x_1(r):\begin{cases} x(0)、\quad x(4) & \text{偶序列} \\ x(2)、\quad x(6) & \text{奇序列} \end{cases}$$

同理：$$x_2(r):\begin{cases} x(1)、\quad x(5) & \text{偶序列} \\ x(3)、\quad x(7) & \text{奇序列} \end{cases}$$

$$\text{设}:\begin{cases} x_1(2l)=x_3(l) & \text{偶序列} \\ x_1(2l+1)=x_4(l) & \text{奇序列} \end{cases} \quad (l=0\ldots N/4-1) \quad \text{此处，} l=0,1$$

$$\text{设}:\begin{cases} x_2(2l)=x_5(l) & \text{偶序列} \\ x_2(2l+1)=x_6(l) & \text{奇序列} \end{cases} \quad (l=0\ldots N/4-1) \quad \text{此处，} l=0,1$$

那么，$X_1(k)$又可表示为：

$$X_1(k)=\sum_{l=0}^{N/4-1}x_1(2l)W_{N/2}^{2lk}+\sum_{l=0}^{N/4-1}x_1(2l+1)W_{N/2}^{(2l+1)k}$$

$$=\sum_{l=0}^{N/4-1}x_3(l)W_{N/4}^{lk}+W_{N/2}^{k}\sum_{l=0}^{N/4-1}x_4(l)W_{N/4}^{lk}$$

$$=X_3(k)+W_{N/2}^{k}X_4(k)$$

$$\left.\begin{aligned}X_1(k)&=X_3(k)+W_{N/2}^{k}X_4(k)\\ X_1(k+N/4)&=X_3(k)-W_{N/2}^{k}X_4(k)\end{aligned}\right\}\quad k=0,1,\ldots\ldots N/4-1$$

$X_2(k)$也可以进行相同的分解：

$$X_2(k) = \sum_{l=0}^{N/4-1} x_2(2l) W_{N/2}^{2lk} + \sum_{l=0}^{N/4-1} x_2(2l+1) W_{N/2}^{(2l+1)k}$$

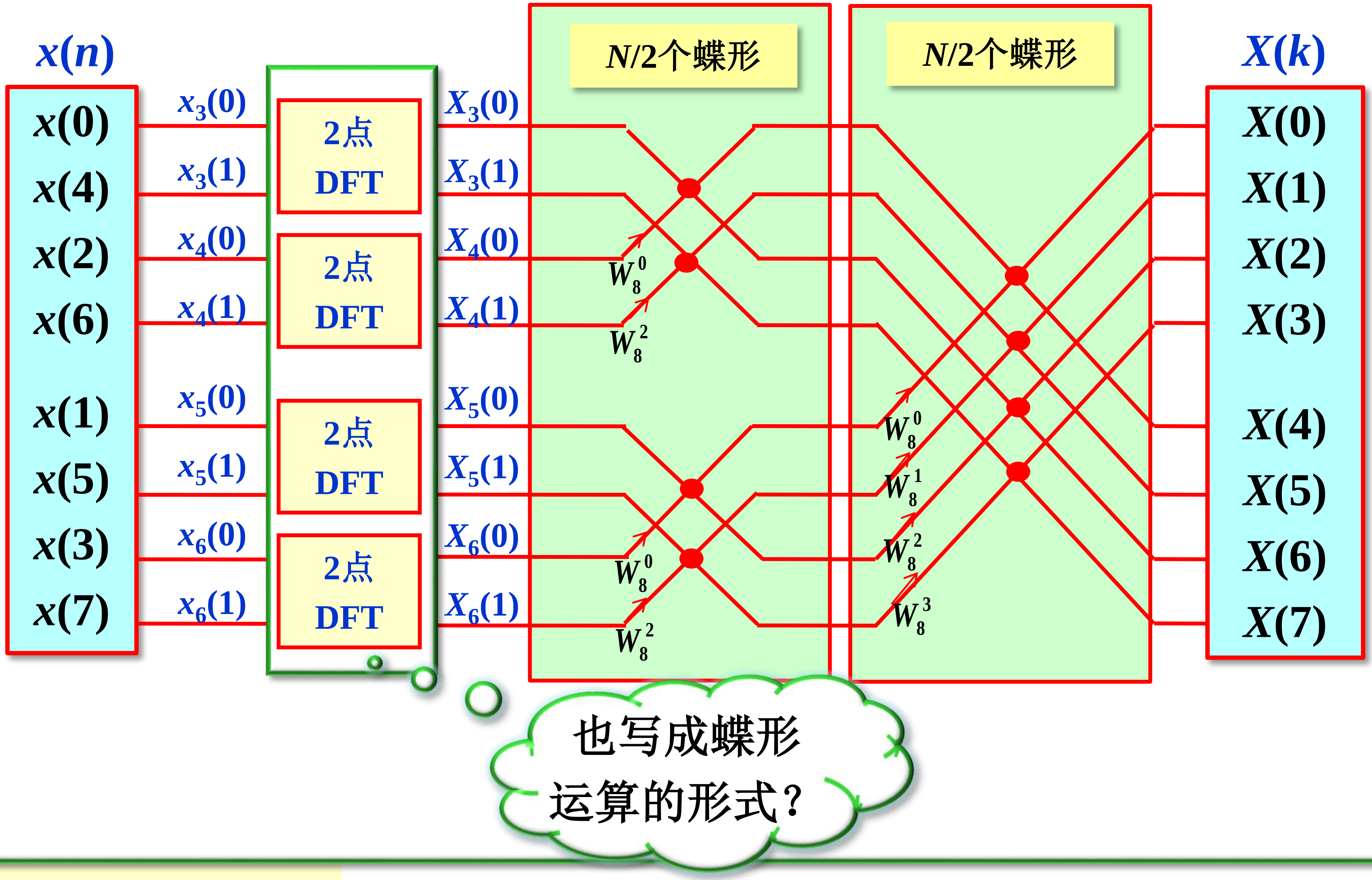
$$= \sum_{l=0}^{N/4-1} x_5(l) W_{N/4}^{lk} + W_{N/2}^{k} \sum_{l=0}^{N/4-1} x_6(l) W_{N/4}^{lk}$$

$$= X_5(k) + W_{N/2}^{K} X_6(k)$$

$$\left.\begin{aligned} X_2(k) &= X_5(k) + W_{N/2}^{k} X_6(k) \\ X_2(k + N/4) &= X_5(k) - W_{N/2}^{k} X_6(k) \end{aligned}\right\} \quad k = 0,1,\ldots\ldots N/4 - 1$$

注意：通常我们会把 $W_{N/2}^{k}$ 写成 W_{N}^{2k}。

N点DFT的二次时域抽取分解图及运算量(N=8)

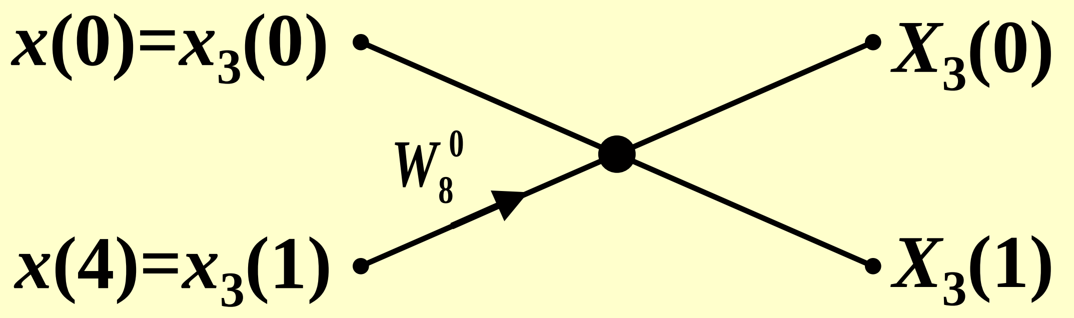


4.2 基于时间抽取的基-2-FFT快速算法

$$X_3(k) = \mathrm{DFT}[x_3(l)] = \sum_{l=0}^{N/4-1} x_3(l) W_{N/4}^{kl} = x_3(0) + W_2^k x_3(1)$$

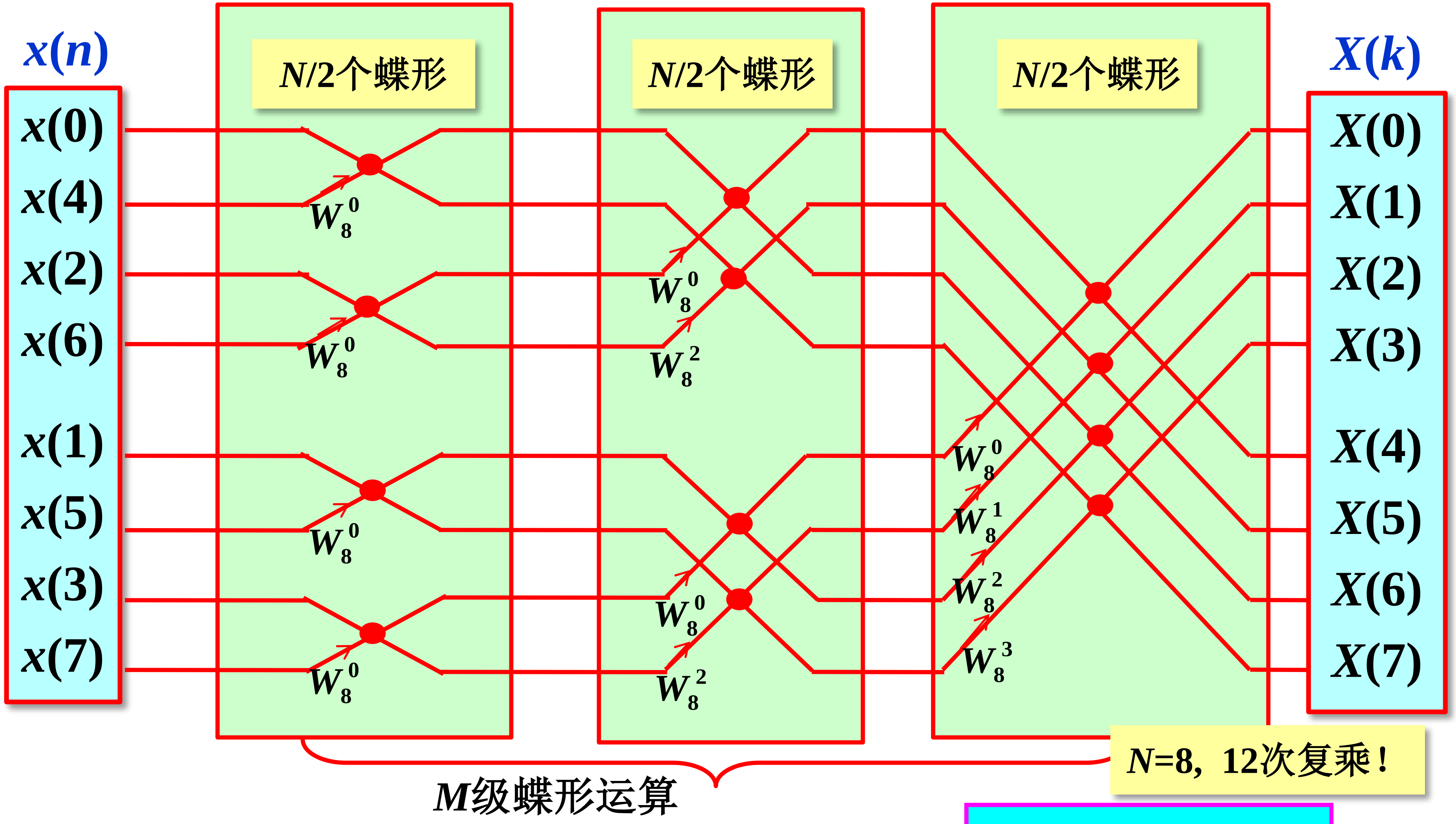
$$X_3(0) = \underline{x_3(0) + W_8^0 x_3(1)}$$

$$X_3(1) = x_3(0) + W_2^1 x_3(1) = \underline{x_3(0) - W_8^0 x_3(1)}$$



基2-DIT-FFT算法 蝶形流图（N=8，M=3）

$M = \log_2^N$



基2-DIT-FFT算法复数乘法次数：

$$\frac{N}{2}\cdot M = \frac{N}{2}\log_2^N$$

三、线性卷积和FFT方法求解LSI系统输出的运算量对比

- 时域直接求解 → 乘法次数：$N_1 \times N_2$ → 62500次

N_1=250，N_2=250

$$y_l(n) = x(n) * h(n) = \sum_{m=-\infty}^{\infty} x(m)h(n-m)$$

- FFT求解法 → 乘法次数：$3\frac{N}{2}\log_2^N + N$ → 7424次

$x(n)$（N_1点）→ 补零至N点 → N点FFT → $X(k)$

$h(n)$（N_2点）→ 补零至N点 → N点FFT → $H(k)$

$X(k) \otimes H(k)$ → $Y(k)$ → N点IFFT（N=512）

$N \geq N_1$+N_2−1

$$y(n) = \mathrm{IDFT}[X(k)\cdot H(k)] = x(n) \, N \, h(n) = y_l(n)$$

第四章　快速傅里叶变换

Fast Forurier Transform

4.2　基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法原理（2）

华东理工大学信息科学与工程学院 万永菁

第四章　快速傅里叶变换

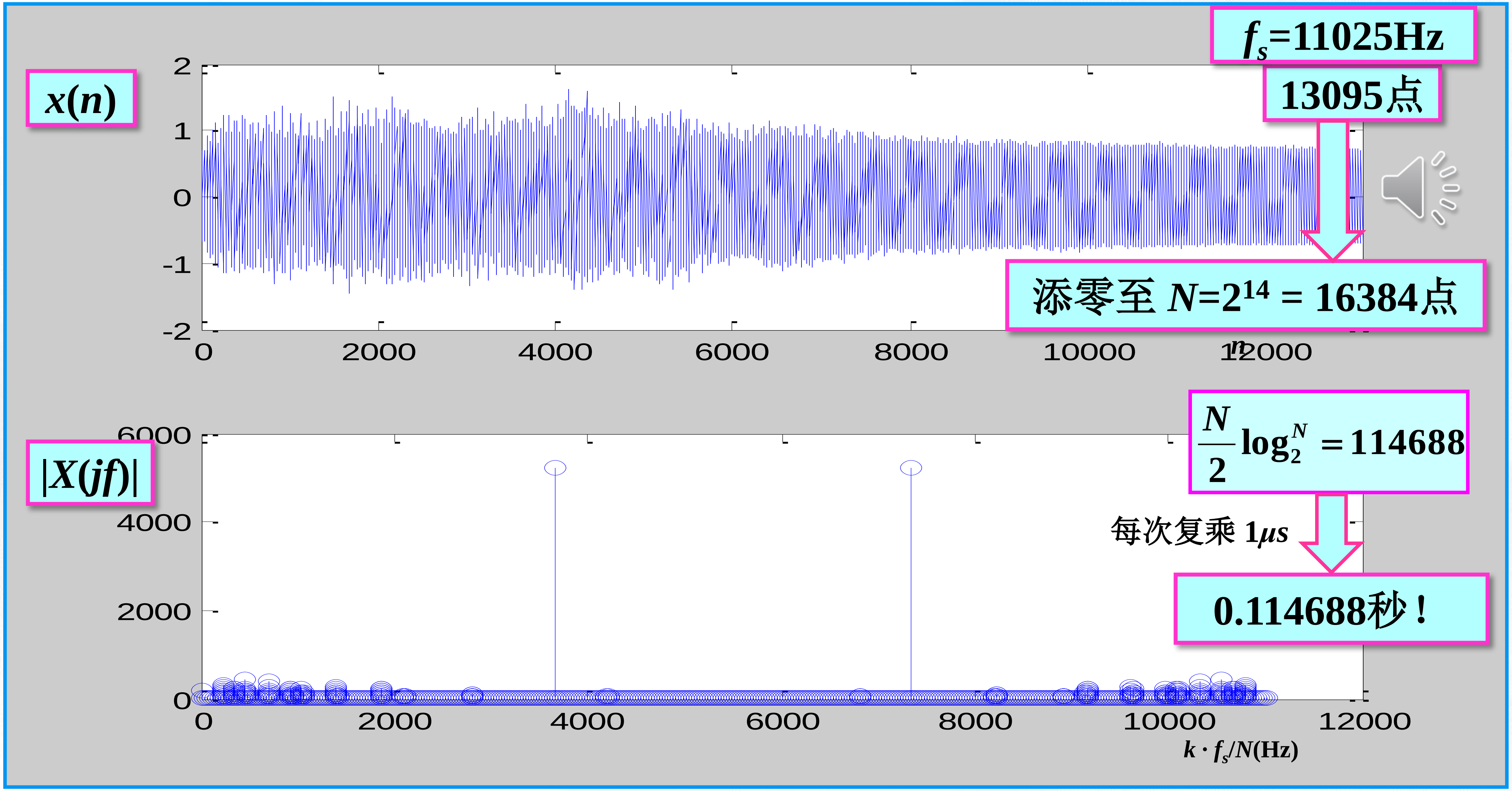
Fast Forurier Transform

4.2 基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法 实例分析

华东理工大学信息科学与工程学院 万永菁

➢ 再探仿真实例：用基2-DIT-FFT方法分析含噪音频的频谱



例、如果通用计算机的速度为平均每次复乘需要5μs，每次复加需要0.5μs，用它来计算512点的DFT[$x(n)$]，问：

(1) 直接用DFT计算需要多少时间？

(2) 用FFT需要多少时间？

解：(1) 直接用DFT进行计算：

复乘：$T_1=N^2\times5\times10^{-6}=1.31072$秒

复加：$T_2=N(N-1)\times0.5\times10^{-6}=0.130816$秒

$T=T_1+T_2=$**1.441536**秒

(2) 用FFT进行运算：

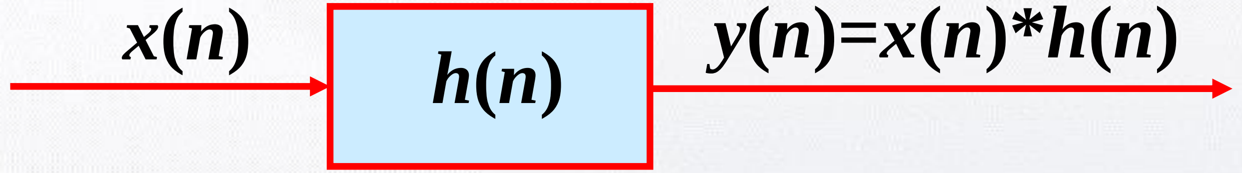
复乘：$T_1'=(N/2)\log_2^N\times5\times10^{-6}=0.01152$秒

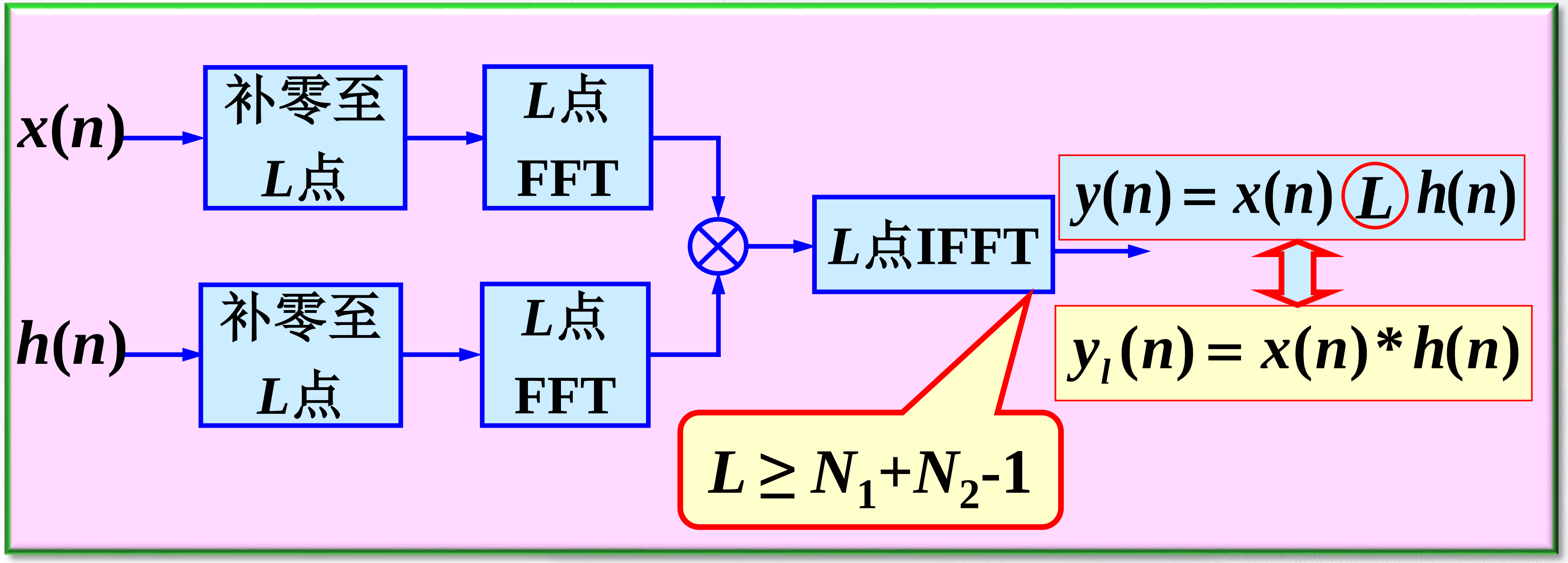
复加：$T_2'=N\log_2^N\times0.5\times10^{-6}=0.002304$秒

$T'=T_1'+T_2'=$**0.013824**秒

4.2 基于时间抽取的基-2-FFT快速算法

例、长度为240点的序列$x(n)$与长度为N点的$h(n)$卷积。当N=10和240时，直接进行卷积 $x(n)*h(n)$ 和用 IFFT[$X(k)\cdot H(k)$] 的方法相比，那种方法求解$y(n)$的效率更高？





直接进行卷积（N=10）：

$$y(n)=x(n)*h(n)=\sum_{n=-\infty}^{\infty}x(n)h(n-m)$$

乘法次数：240×10 = 2400次

用FFT的方法（N=10）：添零到256点，L=256

乘法次数：

$3\times(L/2)\log_2{}^{L}+L = 3\times128\times8+256 = 3328$次

直接进行卷积（N=240）：

$$y(n)=x(n)*h(n)=\sum_{n=-\infty}^{\infty}x(n)h(n-m)$$

乘法次数：240×240 = 57600次

用FFT的方法（N=240）：添零到512点，L=512

乘法次数：

$$3\times(L/2)\log_2 L + L = 3\times256\times9+512 = 7424\text{次}$$

第四章　快速傅里叶变换

Fast Forurier Transform

4.2 基于时间抽取的基-2-FFT快速算法

基于时间抽取的基-2-FFT快速算法 实例分析

华东理工大学信息科学与工程学院 万永菁